



ReSIST: Resilience for Survivability in IST

A European Network of Excellence

Contract Number: 026764

Deliverable D12

Resilience-Building Technologies: State of Knowledge

Appendices – Papers produced by ReSIST partners since January 2006

Report Preparation Date: September 2006

Classification: Public Circulation

Contract Start Date: 1st January 2006

Contract Duration: 36 months

Project Co-ordinator: LAAS-CNRS

Partners: Budapest University of Technology and Economics
City University, London
Technische Universität Darmstadt
Deep Blue Srl
Institut Eurécom
France Telecom Recherche et Développement
IBM Research GmbH
Université de Rennes 1 – IRISA
Université de Toulouse III – IRIT
Vytautas Magnus University, Kaunas
Universidade de Lisboa
University of Newcastle upon Tyne
Università di Pisa
QinetiQ Limited
Università degli studi di Roma "La Sapienza"
Universität Ulm
University of Southampton



Information Society
Technologies



SIXTH FRAMEWORK PROGRAMME

ReSIST D12 - APPENDIX CONTENTS:

Part ARCH

[Arief et al. 2006] B. Arief, A. Iliasov, and A. Romanovsky, "On Using the CAMA Framework for Developing Open Mobile Fault Tolerant Agent Systems", Workshop on Software Engineering for Large-Scale Multi-Agent Systems, pp. 29-35, May 2006.

[Avizienis 2006] A. Avizienis. "An Immune System Paradigm for the Assurance of Dependability of Collaborative Self-Organizing Systems", Proceedings of the IFIP 19th World Computer Congress, 1st IFIP International Conference on Biologically Inspired Computing, pp. 1-6., 2006.

[Becker et al. 2006] S. Becker, A. Brogi, I. Gorton, S. Overhage, A. Romanovsky and M. Tivoli, "Towards an Engineering Approach to Component Adaptation", R. H. Reussner, J. A. Stafford and C. A. Szyperski, editors, Architecting Systems with Trustworthy Components, Vol. 3938 of LNCS, pp. 193-215, 2006.

[Damasceno et al. 2006] K. Damasceno, N. Cacho, A. Garcia, A. Romanovsky, and C. Lucena, "Context-Aware Exception Handling in Mobile Agent Systems: The MoCA Case", Workshop on Software Engineering for Large-Scale Multi-Agent Systems, May 2006.

[Gashi and Popov 2006] I. Gashi and P. Popov. "Rephrasing Rules for Off-The-Shelf SQL Database Servers", Proceedings of the 6th European Dependable Computing Conference, October 2006.

[Gashi et al. 2006b] I. Gashi, P. Popov and L. Strigini "Fault Tolerance via Diversity for Off-the-shelf Products: A Study with SQL Database Servers", manuscript, 2006.

[Gonczy and Varro 2006] L. Gonczy and D. Varro "Modeling of Reliable Messaging in Service Oriented Architectures", Andrea Polini, editor, Proceedings of the International Workshop on Web Services Modeling and Testing, pp. 35-49, 2006.

[Karjoth et al. 2006] G. Karjoth, B. Pfitzmann, M. Schunter, and M. Waidner "Service-oriented Assurance-Comprehensive Security by Explicit Assurances", Proceedings of the 1st Workshop on Quality of Protection, LNCS, to appear in 2006.

[Martin-Guillerez et al. 2006] D. Martin-Guillerez, M. Banâtre and P. Couderc, "A Survey on Communication Paradigms for Wireless Mobile Appliances", INRIA Report, May 2006.

[Mello et al. 2006] E. Ribeiro de Mello, S. Parastatidis, P. Reinecke, C. Smith, A. van Moorsel, and J. Webber "Secure and Provable Service Support for Human-Intensive Real-Estate Processes", Proceedings of 2006 IEEE International Conference on Services Computing, Chicago, Illinois, September 2006, p495-502. *[This work won FIRST PRIZE in the IEEE International Services Computing Contest, September 2006].*

[Mian et al. 2006] A. Mian, R. Beraldi, and R. Baldoni, "Survey of Service Discovery Protocols in Mobile Ad Hoc Networks", Technical Report - Midlab 7/06, Dip. Informatica e Sistemistica "Antonio Ruberti", Università di Roma "La Sapienza", 2006.

[Salatge and Fabre 2006] N. Salatge and J.-C. Fabre, "A Fault Tolerance Support Infrastructure for Web Services based Applications", LAAS Research Report No. 06365, May 2006.

[Stankovic and Popov 2006] V. Stankovic and P. Popov, "Improving DBMS Performance through Diverse Redundancy", Proceedings of the 25th International Symposium on Reliable Distributed Systems, October 2006.

[Verissimo et al. 2006] P. Verissimo, N. Neves, C. Cachin, J. Poritz, D. Powell, Y. Deswarte, R. Stroud, and I. Welch, "Intrusion-Tolerant Middleware: The Road to Automatic Security", IEEE Security & Privacy, Vol. 4, No. 4, pp. 54-62, July/August 2006.

Part ALGO

[Baldoni et al. 2006] R. Baldoni, S. Bonomi, L. Querzoni, A. Rippa, S. Tucci Piergiovanni, A. Virgillito, "Fighting Erosion in Dynamic Large-Scale Overlay Networks", Technical Report - Midlab 9/06 , Dip. Informatica e Sistemistica "Antonio Ruberti", Universit di Roma "La Sapienza", 2006.

[Baldoni et al. 2006-07] R. Baldoni, S. Bonomi, L. Querzoni, A. Rippa, S. Tucci Piergiovanni and A. Virgillito, "Evaluation of Unstructured Overlay Maintenance Protocols under Churn", IWDDS 2006 co-located with ICDCS2006.

[Baldoni et al. 2006-10] R. Baldoni, M. Malek, A. Milani, S. Tucci Piergiovanni, "Weakly- Persistent Causal Objects In Dynamic Distributed Systems", To appear in proc. of SRDS 2006, october 2006, Leeds (UK).

[Baldoni et al. 2006-11] R. Baldoni, R. Guerraoui, R. Levy, V. Quema, S. Tucci Piergiovanni, "Unconscious Eventual Consistency with Gossips", To appear in Proc. of SSS 2006, November 2006, Dallas (USA).

[Correia et al., 2006a] Correia, M., Bessani, A. N., Neves, N. F., Lung, L. C., and Ver'issimo, P. (2006a). Improving byzantine protocols with secure computational components. In the report.

[Courtés et al., 2006] Courtés, L., Killijian, M.-O., and Powell, D. (2006). Storage tradeoffs in a collaborative backup service for mobile devices. In Proceedings of the 6th European Dependable Computing Conference (EDCC-6), number LAAS Report #05673, pages 129–138, Coimbra, Portugal.

[Mostefaoui et al. 2006] Mostéfaoui A., Raynal M. and Travers C., Exploring Gafni's reduction and: from Omega-k to wait-free (2p-p/k)-renaming via set agreement. Proc. 20th Symposium on Distributed Computing (DISC'06), Springer Verlag LNCS #4167, pp. 1-15, Stockholm (Sweden), 2006.

[Raynal and Travers 2006] Raynal M. and Travers C., In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement. Invited paper. Proc. 12th Int'l Conference on Principles of Distributed Systems, (OPODIS'06), To appear in Springer Verlag LNCS, 2006.

[Ryan and Schneider 2006] P.Y.A. Ryan and S. A. Schneider, Prêt à Voter with Re-encryption Mixes, School of Computing Science Technical Report CS-TR: 956, Newcastle University, 2006.

Part SOCIO

[Alberdi et al. 2006] Alberdi, E, Ayton, P, Povyakalo, A. A, and Strigini, L. "Automation Bias in Computer Aided Decision Making in Cancer Detection: Implications for System Design". Technical Report, CSR, City University, 2006. 2006.

[Barboni et al. 2006a] Barboni, E, Conversy, S, Navarre, D, and Palanque, P. "Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification". Proceedings of the 13th

conference on Design Specification and Verification of Interactive Systems (DSVIS 2006). 2006. Lecture Notes in Computer Science, Springer Verlag.

[Barboni et al. 2006b] Barboni, E, Navarre, D, Palanque, P, and Basnyat, S. "Exploitation of Formal Specification Techniques for ARINC 661 Interactive Cockpit Applications". Proceedings of HCI aero conference, (HCI Aero 2006). 2006.

[Basnyat and Palanque 2006] Basnyat, S and Palanque, P. "A Barrier-based Approach for the Design of Safety Critical Interactive Application". ESREL 2006 Safety and Reliability for Managing Risk. Safety and Reliability Conference. 2006. Balkema (Taylor & Francis).

[Basnyat et al. Submitted] Basnyat, S, Schupp, B, Palanque, P, and Wright, P. "Formal Socio-Technical Barrier Modelling for Safety-Critical Interactive Systems Design". Special Issue of Safety Science Journal. Submitted.

[Bryans et al. 2006] Bryans, J. W, Ryan, P. Y. A, Littlewood, B, and Strigini, L. "E-voting: dependability requirements and design for dependability". First International Conference on Availability, Reliability and Security (ARES'06). 988-995. 2006.

[Harrison and Loer 2006] Harrison, M. D and Loer, K. "Time as a dimension in the design and analysis of interactive systems". (in preparation).

[Harrison et al. 2006] Harrison, M. D, Campos, J. C, Dohery, G, and Loer, K. "Connecting rigorous system analysis to experience centred design". Workshop on Software Engineering Challenges for Ubiquitous Computing. 2006.

[Palanque et al. 2006] Palanque, P, Bernhaupt, R, Navarre, D, Ould, M, and Winckler, M. "Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification". Ninth International Conference on Space Operations, Rome, Italy, June 18-22, 2006.

[Schupp et al. 2006] Schupp, B, Basnyat, S, Palanque, P, and Wright, P. A Barrier-Approach to Inform Model-Based Design of Safety-Critical Interactive Systems. 9th International Symposium of the ISSA Research Section Design process and human factors integration: Optimising company performances. 2006.

[Sujan and Harrison 2006] Sujan, M and Harrison, M. D. "Investigation of structural properties of hazard mitigation arguments". Analysis of the structure of mitigation arguments and the role of barriers or defences with particular reference to the EUROCONTROL Reduced Vertical Separation Minima Functional Hazard Analysis. 2006.

[Sujan et al. 2006a] Sujan, M, Harrison, M. D, Steven, A, Pearson, P. H, and Vernon, S. J. "Demonstration of Safety in Healthcare Organisations". Proceedings SAFECOMP. Springer LNCS. 2006.

Part EVAL

[Alata et al. 2006] E. Alata, V. Nicomette, M. Kaaniche and M. Dacier, "Lessons learned from the deployment of a high-interaction honeypot", LAAS_Report 06-331, April 2006. To appear in Proc. Sixth European Dependable Computing Conference (EDCC-6), Coimbra, Portugal, October 18-20, 2006.

[Albinet et al. 2007] A. Albinet, J. Arlat and J.-C. Fabre, “Robustness of the Device Driver-Kernel Interface: Application to the Linux Kernel”, LAAS_Report 06-351, May 2006. To appear in *Dependability Benchmarking of Computer Systems*, (K. Kanoun and L. Spainhower, Eds.), IEEE CS Press, 2007.

[Gönczy et al. 2006] L. Gönczy, S. Chiaradonna, F. Di Giandomenico, A. Pataricza, A. Bondavalli, and T. Bartha, “Dependability evaluation of web service-based processes”. In Proc. of European Performance Engineering Workshop (EPEW 2006), LNCS Vol. 4054, pp. 166-180, Springer, 2006.

[Kaâniche et al. 2006] M. Kaâniche, E. Alata, V. Nicomette; Y.Deswarte, M. Dacier, “Empirical analysis and statistical modeling of attack processes based on honeypots” WEEDS 2006 - workshop on empirical evaluation of dependability and security (in conjunction with the international conference on dependable systems and networks, (DSN2006), Philadelphia (USA), June 25 - 28, 2006, pp. 119-124.

[Kanoun and Crouzet 2006] K. Kanoun and Y. Crouzet, “Dependability Benchmarks for operating Systems”, International Journal of Performability Engineering, Vol. 2, No. 3, July 2006, 275-287.

[Kanoun et al. 2007] K. Kanoun, Y. Crouzet, A. Kalakech and A.-E. Rugina, “Windows and Linux Robustness Benchmarks With Respect to Application Erroneous Behavior”, LAAS report, May 2006. To appear in “Dependability Benchmarking of Computer Systems”, (K. Kanoun and L. Spainhower, Eds.), IEEE CS Press, 2007.

[Lamprecht et al. 2006] C. Lamprecht, A. van Moorsel, P. Tomlinson and N. Thomas, “Investigating the Efficiency of Cryptographic Algorithms in Online Transactions,” International Journal of Simulation: Systems, Science and Technology, UK Simulation Society, Vol. 7, Issue 2, pp. 63—75, 2006.

[Littlewood & Wright 2006] B. Littlewood and D. Wright, “The use of multi-legged arguments to increase confidence in safety claims for software-based systems: a study based on a BBN of an idealised example”, 2006.

[Lollini et al. 2006] P. Lollini, A. Bondavalli, and F. Di Giandomenico, “A general modeling approach and its application to a UMTS network with soft-handover mechanism”, Technical Report RCL060501, University of Firenze, Dip. Sistemi e Informatica, May 2006.

[Rugina et al. 2006] A.-E. Rugina, K. Kanoun and M. Kaâniche, “A System Dependability Modeling Framework using AADL and GSPNs”, LAAS-CNRS Report N° 05666, April 2006.

[Salako & Strigini 2006] K. Salako and L. Strigini, “Diversity for fault tolerance: effects of "dependence" and common factors in software development", Centre for Software reliability, City University, DISPO project technical report KS DISPO5 01, Sept 2006.

Part VERIF

[Backes et al. 2006b] M. Backes, B. Pfitzmann, and M. Waidner, “Non-Determinism in Multi-Party Computation”; Workshop on Models for Cryptographic Protocols (MCP 2006), Aarhus, July-August 2006; abstracts as report of ECRYPT (European Network of Excellence in Cryptology, IST-2002-507932).

[Backes et al. 2006c] M. Backes, B. Pfitzmann, and M. Waidner, “Soundness Limits of Dolev-Yao Models”; Workshop on Formal and Computational Cryptography (FCC 2006), Venice, July 2006 (no formal proceedings).

[Micskei and Majzik 2006] Z. Micskei and I. Majzik, “Model-based Automatic Test Generation for Event-Driven Embedded Systems using Model Checkers,” in Proc. of Dependability of Computer Systems (DepCoS '06), Szklarska Poręba, Poland, pp.192-198, IEEE CS Press, 2006.

[Micskei et al. 2006] Z. Micskei, I. Majzik and F. Tam, “Robustness Testing Techniques For High Availability Middleware Solutions,” in Proc. Int. Workshop on Engineering of Fault Tolerant Systems (EFTS 2006), Luxembourg, Luxembourg, 12 - 14 June, 2006.

[Pfeifer and von Henke 2006] H. Pfeifer and F. von Henke, “Modular Formal Analysis of the Central Guardian in the Time-Triggered Architecture”, Reliability Engineering & System Safety, Special Issue on Safety, Reliability and Security of Industrial Computer Systems, Elsevier Ltd., 2006, to appear.

[Serafini et al. 2006] M. Serafini, P. Bokor and N. Suri, “On Exploiting Symmetry to Verify Distributed Protocols”, Fast Abstract, International Conference on Dependable Systems and Networks (DSN) 2006.

[Waeselynck et al. 2006] H. Waeselynck, P. Thévenod-Fosse and O. Abdellatif-Kaddour, “Simulated annealing applied to test generation: landscape characterization and stopping criteria”, to appear in *Empirical Software Engineering*, 2006.

Part Arch – APPENDIX

(Resilience Architecting and Implementation Paradigms)

On Using the CAMA Framework for Developing Open Mobile Fault Tolerant Agent Systems

Budi Arief, Alexei Iliasov and Alexander Romanovsky
School of Computing Science
University of Newcastle upon Tyne
Newcastle upon Tyne NE1 7RU, England

{L.B.Arief, Alexei.Iliasov, Alexander.Romanovsky}@newcastle.ac.uk

ABSTRACT

The paper introduces the CAMA (*C*ontext-*A*ware *M*obile *A*gents) framework intended for developing large-scale mobile applications using the agent paradigm. CAMA provides a powerful set of abstractions, a supporting middleware and an adaptation layer allowing developers to address the main characteristics of the mobile applications: openness, asynchronous and anonymous communication, fault tolerance, device mobility. It ensures recursive system structuring using location, scope, agent and role abstractions. CAMA supports system fault tolerance through exception handling and structured agent coordination. The applicability of the framework is demonstrated using an ambient lecture scenario – the first part of an ongoing work on a series of ambient campus applications.

Categories and Subject Descriptors: D.1 Programming Techniques, D.2.3 Coding Tools and Techniques, D.4.5 Reliability

General Terms: Design, Reliability

Keywords: Mobile computing, fault tolerance, agent systems, coordination, scopes, exception handling, ambient lecture

1. INTRODUCTION

Although the mobile agent paradigm supports structuring systems using decentralised and distributed entities cooperating to achieve their individual aims and promotes system openness, flexibility and scalability, the existing frameworks for development of such systems do not provide adequate means for achieving fault tolerance. The main difficulties here are caused by agent mobility, autonomy and asynchronous communication, system openness and dynamicity, which create new challenges for ensuring system fault tolerance.

In this work, we are focusing on *coordination mobile environments*, which have become very popular in developing mobile agent applications. These environments rely on

the Linda approach to coordination of distributed processes. Linda [7] provides a set of language-independent coordination primitives that can be used for communication-between and coordination-of several independent pieces of software. Linda is now becoming the core component of many mobile software systems because it fits in nicely with the main characteristics of mobile systems. Linda coordination primitives support effective inter-process coordination by allowing processes to put *tuples* in a tuple space shared by these processes, get *tuples* out if they match the requested types, and test for them. A tuple is a vector of typed data values, some of which can be empty, in which case they match any value of a given type. Certain operations, like *get* (or *in*) and *test* (or *inp*), can be blocking.

A number of Linda-based mobile coordination systems have been developed in the last years (including Klaim [3], TuCSon [14] and Lime [15]). Lime is one of the most developed, supported and widely-used examples of such environments. It supports both *physical mobility*, such as a device with a running application travelling along with its user across network boundaries, and *logical mobility*, when a software application changes its platform and resumes execution in a new one. To do that, Lime employs a distributed tuple space. Each agent has its own persistent tuple space that physically or logically moves with it. When an agent is in a location where there are other agents or where there is a network connectivity to other Lime hosts, a new shared tuple space can be created, thus allowing agents to communicate. If connection is lost or some agents leave, parts of the shared tuple space became inaccessible. Lime middleware – implemented in Java – hides all the details and complexities of the distributed tuple space control and allows agents to treat it as normal tuple space using conventional Linda operations.

Exception handling [4] is widely accepted to be the most general approach to ensuring fault tolerance of complex applications facing a broad range of faults. It provides a sophisticated set of features for developing effective fault tolerance using handlers specially tailored for the specific exception and system state in which the error is detected. It ensures nested system structuring and separates normal system behaviour from the abnormal one. Our analysis [12] shows that the existing Linda-based mobile environments do not provide sufficient support for development of fault tolerant mobile agent systems. The real challenge here is to develop general mechanisms that smoothly combine Linda-based mobility with exception handling. The two key features of mobile agents are asynchronous communication

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SELMAS'06, May 22–23, 2006, Shanghai, China.

Copyright 2006 ACM 1-59593-085-X/06/0005 ...\$5.00.

and agent anonymity. This is what makes mobile agents such a flexible and powerful software development paradigm. However, traditional fault tolerance and exception handling schemes are not directly applicable in such environments.

In this paper, we discuss a novel framework for disciplined development of open fault tolerant mobile agent systems and show how it is being applied in developing an ambient campus application. This framework offers a set of powerful abstractions to help developers by supporting exception handling, system structuring and openness. These abstractions are supported by an effective and easy-to-use middleware which ensures high system scalability and agent compatibility. The plan of the paper is as follows. In the next section we introduce our CAMA framework in detail by describing the main abstractions offered to system developers, a novel exception handling mechanism and our current work on CAMA implementation. This is followed by a section discussing our experience in applying CAMA in the development an ambient lecture scenario as a part of our ongoing work on ambient campus applications. The last section of the paper outlines our plans for the future work.

2. CONTEXT AWARE MOBILE AGENTS

We have developed a framework called CAMA (*Context-Aware Mobile Agents*), which encourages disciplined development of open fault tolerant mobile agent applications by supporting a set of abstractions ensuring exception handling, system structuring and openness. These abstractions are backed by an effective and easy-to-use middleware allowing high system scalability and guaranteeing agent compatibility.

2.1 CAMA Abstractions

Any CAMA system consists of a set of *locations*. A location is a container for *scopes*. A scope provides a coordination space within which compatible agents can interact using the scoping mechanism described below. *Agents* are the active entities of the system. An agent is a piece of software that conforms to some formal *specification*. Each agent is executed on a *platform*; several agents may reside on a single platform. A platform provides an execution environment for agents as well as an interface to the location middleware. Figure 1 shows how these abstractions are linked.

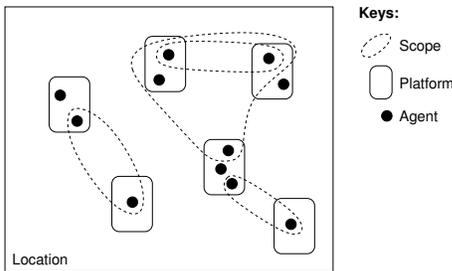


Figure 1: Location, scopes, platforms and agents in Cama

An agent is built using one or more *roles*. A role is a specification of one specific functionality of an agent. A composition of all agent roles forms its specification.

Location can be associated with a particular physical location (such as lecture theatre, warehouse or meeting room)

and can have certain restrictions on the types of supported scopes. Location is the core part of the system as it provides means of communication and coordination among agents. We assume that each location has a unique name. This roughly corresponds to the IP address of the host in a network (which are usually unique) on which it resides. A location must keep track of the agents present and their properties in order to be able to automatically create new scopes and restrict access to the existing ones. Locations may provide additional services that can vary from one instance to another. These are made available to agents within what appears to be a normal scope where some of the roles are implemented by the location system software. As with all the scopes, agents are required to implement specific roles in order to connect to a location-provided scope. Few examples of such services include printing on a local printer, accessing the internet, making a backup to a location storage, and migrating to another location.

Agent *context* represents the circumstances in which an agent find itself [17]. Generally speaking, a context includes all information from an agent environment which is relevant to its activity. The context of an agent in CAMA consists of the following parts: the state connections to the engaged locations; the names, types and states of all the visible scopes in the engaged locations; and the state of scopes in which the agent is currently participating, including the tuples contained in these scopes. A set of all locations defines global structuring of the agent context. This context changes when an agent migrates from one location to another.

Agents represent the basic structuring unit in CAMA applications. To deal with various functionalities that any individual agent provides, CAMA introduces agent role as a finer unit of code structuring. A role is a structuring unit of an agent, and being an important part of the scoping mechanism, it allows dynamic composition of multi-agent applications, as well as being used to ensure agent interoperability and isolation.

Scope structures the activity of several agents in a specific location by dynamically encapsulating roles of these agents. Scope also provides an isolation of several communicating agents thus structuring the communication space.

A set of agents playing different roles can dynamically instantiate a multi-agent application. A simple example is a client-server model where a distributed application is constructed when agents playing two roles meet and collaborate. An agent can have several roles and use them in different scopes. A server agent can provide the same service in many similar scopes. In addition it can also implement a client role and act as a client in some other scopes.

Supporting system openness is one of the top design objectives of CAMA. Openness is understood here as the ability to create distributed applications composed of agents developed independently. To this end CAMA provide powerful abstractions that help to dynamically compose applications from individual agents, an agent isolation mechanism and service discovery based on the scoping mechanism.

Scoping mechanism

The CAMA agents can cooperate only when they participating in the same scopes. This abstraction is supported by a special construct of coordination space called *scope*. Scoping is a means to *structure* agent activity by arranging agents into groups according to their intentions. Scoping

also allows agent communication to be configured to meet to the requirements of the individual groups. Reconfigurations happen automatically, thus allowing agents (and their developers) to focus solely on collaboration with other agents participating in the same scope. There are several benefits of agent system structuring using scopes:

- scopes provide higher-level abstractions of communication structuring;
- they reduce the risk of creating ad hoc structures that maybe incorrect, malfunctioning or cyclic;
- this structuring enforces strong relationship among agents supporting interoperability and exception handling;
- scopes support simple semantics thus facilitating formal development;
- scopes become units of fault tolerant system ensuring error confinement and supporting error recovery at the scope level.

A scope is a dynamic data container that provides an *isolated* coordination space for *compatible* agents. This is done by restricting visibility of tuples contained in the scope only to these agents. we say that a set of agents is compatible if there is a composition of their roles that forms an instance of an abstract scope model.

Agents can issue a request to create a scope, and when all the preconditions are satisfied, a scope is atomically instantiated by the hosting location. The scope creation request includes a scope identifier (a string) and a scope requirement structure. The request returns the name of a newly created scope. The agent creating the scope can use it to join the scope, to make it public (visible to other agents), to leave it and to remove it.

Scope has a number of attributes divided into two categories: scope *requirements* and scope *state*. Scope requirements essentially define the type of a scope, or, in other words, the kind of activities supported by it. Scope requirements are derived from a formal model of a scope activity and, together with agent roles, form an instance of the abstract scope model. State attributes characterise a unique scope instance. In addition to these attributes, scope contains *data* represented as *tuples* in the coordination space. Along with these data, there may be *subscopes* which define *nested activities* that may happen inside of the scope.

Nested scopes are used to structure large multi-agent applications into smaller parts which do not require participation of all agents. Such structuring has a number of benefits. It isolates agents into groups, thus enhancing security. It also links coordination space structuring with activity structuring, which supports localised error recovery and scalability. There is no hard rule when to use nested scopes. However, for reasons stated above, any application incorporating different modes of communication or different types of activities should use subscopes. Online shop is an example of such application. A seller publicly communicate with buyers while the latter are looking around for some products. However, payment must be a private activity involving only the seller and the buyer. In addition to obvious security benefits, a dedicated payment subscope helps to determine which agents must be involved into recovery should a failure happen during payment.

Restrictions on roles dictate the roles that are available in the scope, and how many agents are allowed for any given role. The latter is defined by two numbers: the minimum

number of agents required for a given role and the maximum number of agents allowed for a given role. A *scope-state* tracks the number of currently-taken roles and determines whether the scope is ready for agent collaboration or whether more agents are allowed to join.

The existing scoping mechanisms (e.g. [18, 13]) are not explicitly developed to support data and behaviour encapsulation or isolation crucial for error confining and recovery. None of them is directly applicable for dealing with mobile agents interacting using coordination spaces (see our analysis in [12]). Also, these schemes do not support the set of abstractions which we have identified as crucial for CAMA.

Basic Operations in CAMA

In CAMA, all the communication within a location happens through a single shared tuple space. This leads to asymmetrical design of the middleware where the tuple space operations are implemented in a *location middleware* while agents only carry a lightweight *adaptation layer*. On top of the coordination primitives derived from Linda, the CAMA middleware provides the following operations:

- **engage(id)** - issues a new location-wide name that is unique and unforgeable for agent *id*. This name is used as agent identifier in all other role operations.
- **disengage(a)** - makes issued name *a* invalid.
- **create(a, n, R)@s** ($n \notin 1.s$) - agent *a* creates a new subscope within scope *s* called *n* with given scope requirements *R* at location *1*. The created scope becomes a private scope of agent *a*.
- **delete(a, n)@1.s** ($n \in 1.s \wedge a$ is owner of $1.s.n$) - agent *a* deletes a subscope called *n* contained in scope *s*. This operation always succeeds if the requesting agent is the owner of the scope. If the scope is not in the pending state then all the scope participants shall receive **CamaExceptionNotInScope** exception notifying the scope's closure. This procedure is executed recursively for all the subscopes contained in the scope.
- **join(a, n, r)@s** ($n \in 1.s \wedge r \in n \wedge n$ is pending or expanding) - adds agent *a* into scope *n* contained in *1.s* with role *r*. This operation succeeds if scope *1.s.n* exists and agent *a* is allowed to take the specified role in the scope. This operation may cause the scope to change state.
- **leave(a, n, r)@s** (*a* is in $1.s.n$ with role(s) *r*) - removes agent *a* with roles *r* from scope *1.s.n*. The calling agent must be already participating in the scope. This operation may also change the state of the scope.
- **put(a, n)@s** - agent *a* advertises scope *n* contained in scope *s*, thus making it a public scope. A public scope is visible and accessible by other agents.
- **get(a, r)@s**: enquires the names of the scopes contained in scope *1.s* and supporting role(s) *r*.

An agent always starts its execution by looking for available locations nearby. Once it engages a location it can join a scope or create a new one. An agent needs to know the name of the scope it intends to join. It can be the name of an existing scope or the name of a new scope created by this agent. When joining a scope, an agent specifies its role in the scope. In the current implementation of the middleware, an agent can choose a role in a scope from one of the roles it implements. The **join** operation returns a handle for a scope, which can be used by an agent to collaborate with other agents through Linda coordination primitives. To cre-

ate a scope, an agent must specify the name of the scope and the scope requirements, which define the possible roles within the scope and their restrictions.

Physical and Logical Mobility

Physical mobility allows devices carrying the agent code to move between locations. Logical mobility allows agent code and state to be moved from one location to another.

Physical mobility in CAMA is implemented using connectivity of the devices to the locations. When such a connectivity is established, the agent running on the device receives special event notifying it about discovery of the new location. CAMA allows any agent to access the list of active locations it is connected to at any time. An agent receives a predefined disconnection exception when the connectivity is lost. To support this functionality, the location middleware periodically sends hard beats messages in the proximity.

The CAMA middleware does not support logical mobility as the first class concept since the CAMA architecture does not allow locations to see each other. Nevertheless, agent migration can be provided through the standard inter-agent communication. Data can be moved between locations in CAMA by agents working at both locations at the same time, or by an agent physically migrating between two locations or by using some other capability supporting data transfer between locations. In particular, we have implemented a simple proof-of-concept support ensuring weak code mobility. In this implementation, a dedicated agent provides a service of data transfer between locations using internet or LAN networking. Using this service, any agent can transfer itself or another agent to another location.

2.2 Fault Tolerance

The CAMA framework supports application-level fault tolerance by providing a set of abstractions and a supporting middleware that allow developers to design effective error detection and recovery mechanisms. The main means for implementing fault tolerance in CAMA is a novel exception handling mechanism which associates scopes with the exception contexts. Scope nesting provides recursive system structuring and error confinement. In addition to this, the CAMA middleware supports a number of predefined exceptions (e.g. the connection and disconnection ones, violation of the scope constraints, etc.).

In developing exception handling support for CAMA, we relied on our previous work reported in [12], in which we proposed and evaluated a novel exception handling scheme developed for coordination-based agent applications. Here we give a brief overview of our exception handling mechanism; the full description can be found in [11]. The main novelty of the CAMA mechanism is that it explicitly links nested scopes with the exception contexts.

Exception handling in CAMA allows fast and effective application recovery by supporting flexible choice of the handling scope and of the exception propagation policy. The mechanism of the exception propagation is complimentary to the application-level exception handling. All the recovery actions are implemented by application-specific handlers attached to agents. The ultimate task of the propagation mechanism is to transfer exceptions between agents in a reliable and secure way. However, the freedom of agent behaviour in agent-based systems does not allow any guarantees of reliable exception propagation to be given in a general

case. In particular, the situations can be clearly identified when exceptions may be lost or not delivered within a predictable time period. This is the case for CAMA as well. To alleviate this, for example, in a mobile agent application requiring cooperative exception handling involving several agents, agents behaviour must be constrained in some way to prevent any unexpected migrations or disconnections. In our ongoing work we are developing techniques supporting formal analysis of exception handling behaviour of the multi agent systems.

There are three basic operations available to the CAMA agents for catching and raising inter-agent exceptions. These functionalities are complementary and orthogonal to the application-level mechanism used for programming internal agent behaviour.

The **raise** operation propagates an exception to an agent or a scope. There are two variants of this operation:

- **raise(m, e)** - raises exception **e** as a reaction to message **m**. The message is used to trace the producer and to deliver an exception to it. The operation fails if the destination agent has already left the scope in which the message was produced.
- **raise(s, e)** - raises exception **e** in all participants of scope **s**.

The crucial requirement for the propagation mechanism is to preserve all the essential properties of agent systems such as anonymity, dynamicity and openness. The exception propagation mechanism does not violate the concept of anonymity since we prevent disclosure of agent names at any stage of the propagation process. Note that the **raise** operation does not deal with names or addresses of agents. Moreover, we guarantee that our propagation method cannot be used to learn the names of other agents.

Two other operations, **check** and **wait** are used to explicitly poll and wait for inter-agent exceptions.

- **check** - raises exception **E(e)** if there are any pending exceptions for the calling agent. **E(e)** is a local envelop for the inter-agent exception **e**.
- **wait** - waits until any inter-agent exception appears for the agent and raises it in the same way as the previous operation.

Systematic use of exception handling should allow developers to design mobile agent applications tolerating a *broad range of faults*, including disconnections, agent mismatches, malicious or unanticipated agent activity, violations of system properties, potentially harmful changes in the system environment, reduced amount of resource available, as well as users' mistakes.

Unfortunately, there has not been much work carried out in this area. Paper [19] introduces a guardian model in which each agent has a dedicated guardian responsible for handling all agent exception. This model is general enough to be applied in many types of mobile systems but it does not directly address the specific characteristics of the coordination paradigm. Another relevant work is on exception handling in concurrent object-oriented language Oz [20]. In this system, exceptions can be propagated between the mobile callee and caller objects. The approach proposed is not applicable to the coordination-based mobile systems. Moreover, the main intention behind this work is not to support the development of open dynamic agent applications.

2.3 CAMA Implementation

In the current version of the CAMA system, the location middleware is implemented in C (we call it *cCAMA*). This allows us to achieve the best possible performance of the coordination space and to effectively implement numerous extension, such as the scoping mechanism. The location middleware implementation is quite compact - it consists of approximately 6000 lines of C code and should run on most Unix platforms. We have so far tested it on Linux FC2 and Solaris 10. The full implementation of the location middleware is available at SourceForge [9].

In order to use the location middleware mentioned above, we have developed a CAMA adaptation layer in Java¹ called *jCAMA*. This adaptation layer defines several classes for representing – among others – the abstract notions of Location, Scope and Linda coordination primitives. *jCAMA* provides an interface through which mobile agents or applications can be developed easily.

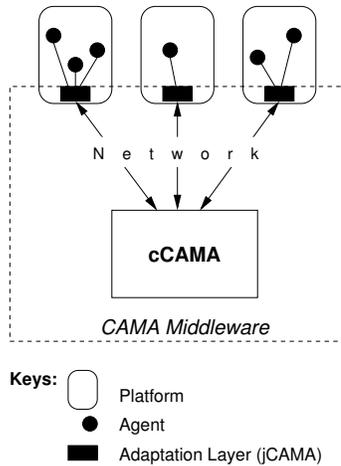


Figure 2: Cama architecture

A diagrammatic representation of the CAMA-based system architecture can be seen in Figure 2. Each platform carries a copy of *jCAMA*. Agents residing on a platform uses the features provided by *jCAMA* to connect over the wireless network to the *cCAMA* location middleware.

It is possible to construct other adaptation layers for different platforms and languages. For now, the *jCAMA* Java adaptation layer outlined above permits agent development for PocketPC-based PDAs. It has a very small footprint (~60Kb) and can be used with both standard Java and J2ME. In the future we plan to develop adaptation layers for other languages such as Python and Visual Basic, as well as versions compatible for smartphone devices.

3. AMBIENT LECTURE APPLICATION

This case study provides a demonstration on how the CAMA framework can be used in developing open, dynamic and pervasive systems involving people carrying hand held devices (e.g. PDAs) to help them in their daily activities.

3.1 Introduction

We focus on the activities performed by students and teachers during a lecture (the *ambient lecture* scenario) and

¹We use Java for developing the applications for PDAs.

consider a set of requirements that define this scenario. This set will be extended to cover more general *ambient campus* scenarios (i.e. location-aware activities that can be performed on campus) such as interactive/smart map, events announcer, library application and students organiser.

There are several other projects aiming to integrate software systems – including mobile applications – into education or campus domain. The ActiveCampus project [8] aims to provide location-based services such as *Map* service (showing outdoor and indoor map of the user’s vicinity along with activities happening there) and *Buddies* service (showing colleagues and their locations, as well as sending messages to them). The ActiveCampus system is implemented as a web server using PHP and MySQL. ActiveClass [16] is a client-server application for encouraging in-class participation using PDAs allowing students to ask questions regarding the lecture in anonymous manner, hence overcoming the problem of shyness among many students.

Gay et. al. carried out an experiment investigating the effects of wireless computing in classroom environment [6]. Students were given laptop computers with wireless or wired connection to the internet, allowing them to use any existing tools and services such as web browsers, word processors, instant messaging software – as well as any additional software they wish to install. The results suggest that the introduction of wireless computing in learning environments can potentially affect the development, maintenance and transformation of learning communities, but not every teaching activity or learning community can or should successfully integrate mobile computing applications.

Classtalk [5] is a classroom communication system that allows teacher to present questions for small group work, collect the answers and display the histograms showing how the class answered those questions. Up to four students can be in one group, sharing one input device (a palmtop), which is wired to the central computer controlled by the teacher.

Similar to Classtalk, our system allows students to be grouped together in order to carry out some task given by the teacher. The novelty of our approach lies in the communication channel (wireless instead of wired connection) as well as in using the framework for supporting scoping and fault tolerance (the mechanisms described in Section 2).

3.2 Traceable Requirements

We started work on the scenario by producing a requirements document [2], which consists of an explanatory text, diagrams, and requirements definitions. The requirements definitions are arranged using a specially-developed taxonomy which allows us to structure them according to various views on system behaviour, including: environment (EN), agent states (ST), service requirements and restrictions (SV), security (SE) and fault tolerance (FT). Each requirement is given a number within the group, for example:

- EN 1: The scenario is composed of users, locations and ambient computing environment (ACE)
- ST 1: The agents’ top-level states are *lecture, free, migrating, outside* and *emergency*
- SV 12: *Teacher distributes lecture material*
- FT 14: Migration activity must tolerate *wireless disconnection and loss of ACE support*

At the high level, the system consists of users (people participating in the scenario, i.e. teachers and students),

locations (rooms with wireless connectivity) and *ambient computing environment* (ACE). ACE is composed of wireless hotspots, software agents and computing platforms (desktop computers or PDAs) on which the agents are run.

The interactions among users are done through agents. Each location provides a CAMA location middleware through which agents exchange information. Agents connect to the location middleware using the wireless hotspot available in each room.

Each teacher and student has an agent associated with him/her and assisting his/her participation in the lecture. During a lecture, teachers and students can be engaged in the following activities: lecture initiation, material dissemination, organisation of students into groups, individual or group student work, and questions and answers session.

3.3 Design

The ambient lecture system is being designed to meet the requirements in [2]. In this design, each classroom is a location with a wireless support, in which a lecture is conducted. An agent can take one of the two roles: teacher or student. The teacher agent runs on a desktop computer available in the classroom, while student agents are executed on PDAs (each student is given a PDA).

We use scoping mechanism described in Section 2.1 to structure the system. The teacher agent creates the outer scope constituting the lecture which student agents join. A lecture starts when there is one teacher agent and a predefined number of student agents joining this scope.

To support better system structuring, data and behaviour encapsulation, as well as fault tolerance, all major activities during the lecture are conducted within subscopes (nested scopes). The group work is one of the activities performed as a nested scope. Teacher – through his/her agent – arranges students into groups, so that only students belonging to the same group can communicate with each other through their agent. Each group is then given a task to solve (could be the same task for all groups). Students within the same group work together on the solution and present their answer at the end of the group work stage.

At the beginning of any lecture, all agents (teacher and students alike) are placed in the main scope. The teacher agent keeps a list of all students joining the lecture, and through the application’s graphical user interface (GUI), the teacher can select which students to be placed within each group. Each group is given a unique name and the groups are mutually exclusive, i.e. a student cannot belong to more than one group. The teacher agent creates a subscope for each group and issues a *StartGroup* tuple to the student agents involved so that they automatically join the subscope they are assigned to. This is achieved by executing the CAMA *JoinScope* operation that uses the group name as a parameter. This structuring guarantees that while within a group, a student can only send messages to other students belonging to the same group, but he/she will also receive any message sent in the main lecture scope. To achieve this, the CAMA middleware creates a separate thread for each role inside a subscope.

Once a group is created, it is represented as a button (containing the names of the students assigned to this group) on the teacher agent’s GUI. Clicking this button ungroups the students and issues a *EndGroup* tuple to the relevant student agents, making them invoke the *LeaveScope* command.

```
try {
    // Connect to the location middleware
    Connection connection = new Connection("Teacher",
        server, portNo);
    Scope lambda = connection.lambda();

    // Create a lecture scope that allows 1 Teacher
    // agent and up to 10 Student agents.
    ScopeDescr sd = new ScopeDescr(2, "lectureScope").
        add(new RoleRest("Teacher", 1,1)).
        add(new RoleRest("Student", 0,10));
    workScope = lambda.CreateScope("lectureScope", sd);

    // Join the scope and make the scope public
    workScope = workScope.JoinScope("Teacher");
    workScope.PutScope();
}
catch(CamaExceptionInvalidReqs e) { ... }
catch(CamaExceptionNoRoles e) { ... }
...
```

Figure 3: Sample code: scope creation by Teacher agent

Following the fault tolerance requirements, the agents handle a number of potentially erroneous conditions. Some of them are detected by the agents themselves, others are detected by the middleware which raises predefined exceptions declared in the signatures of the CAMA operations. One example of these exceptions is the *CamaExceptionNoRights* exception, indicating that the agent concerned has no right to be in a particular scope, hence it cannot send or receive messages from the tuple space.

3.4 Implementation

We developed an application for the group work activity described in section 3.3. There are two sets of agent software: **Teacher** and **Student**. Commands and data are passed as tuples through the tuple space provided by the location middleware.

Each agent runs at least two threads of execution: one thread handles the GUI and provides a means for sending tuples to the tuple space; another thread polls tuples from the tuple space and interprets the command contained in them. More threads are created when subscoping is used, so that an agent can also poll tuples from within the subscopes.

Figure 3 shows a snippet of the code for the **Teacher** agent, demonstrating how the lecture scope is initiated. Agents can join as a **Teacher** or a **Student**. In this example, only one **Teacher** agent is allowed, along with up to ten **Student** agents. An exception will be raised if this restriction is violated.

Figure 4 shows an example interaction among agents in the ambient lecture scenario. There is one **Teacher** agent, shown on the top of Figure 4. There are three **Student** agents: "Alice" (shown on the bottom left, this agent is run from a desktop computer), "Bob" (bottom right, run from a PDA) and "Tom" (not shown). At some stage, the **Teacher** agent places "Alice" and "Tom" into a group. While they are in this group, all messages they send can only be seen by other agents in the same group (group messages are indicated by a (g) in front of them). Teacher can end the group by clicking on the button representing the group (in this case, the "Alice-Bob" button). When this happens,

all students in that group leave the group subscope and the subscope thread of execution terminates, but they all remain connected to the lecture main scope.

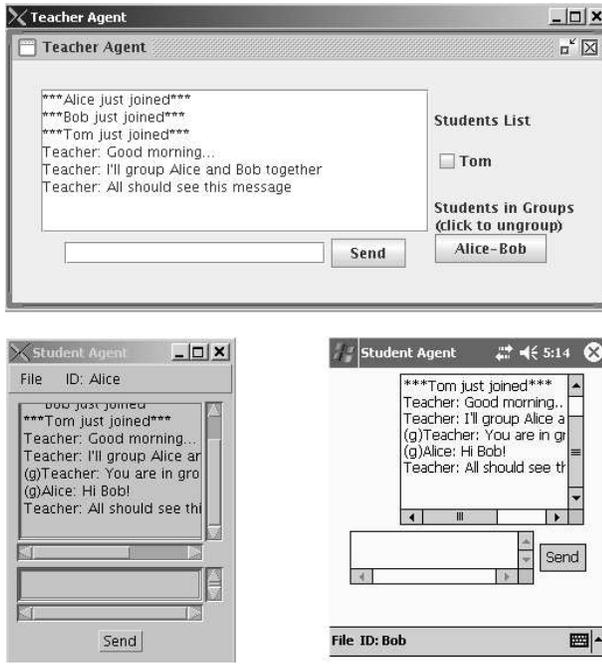


Figure 4: Screen capture of ambient lecture agents

4. FUTURE WORK

Our long-term goal is to support formal development of fault tolerant mobile agent systems. To achieve this goal we are developing a number of formal notations and models defining the CAMA abstractions and the CAMA middleware (some initial results are reported in [10]). We are now working on a top-down design methodology that insures that these systems are correct-by-construction. To ensure the application security, we will use an appropriate encryption mechanism that allows messages to be securely sent between PDAs and the location server. Our other plan is to implement the CAMA location middleware for PDAs to support applications in which locations are physically mobile. In our future work on CAMA for smartphone devices, we will address the facts that smartphones have capabilities that are different from PDAs. For example, smartphones utilise other means for connectivity (such as bluetooth and gprs), which might imply the need to adapt the communication support provided by CAMA.

5. ACKNOWLEDGEMENTS

This work is supported by the IST RODIN Project [1]. A. Iliasov is partially supported by the ORS award (UK).

6. REFERENCES

[1] Rigorous Open Development Environment for Complex Systems. IST FP6 STREP project, <http://rodin.cs.ncl.ac.uk/> [Last accessed: 1 Feb 2006].
 [2] B. Arief, J. Coleman, A. Hall, A. Hilton, A. Iliasov, I. Johnson, C. Jones, L. Laibinis, S. Leppanen, I. Oliver, A. Romanovsky,

C. Snook, E. Troubitsyna, and J. Ziegler. Rodin Deliverable D4: Traceable Requirements Document for Case Studies. Technical report, Project IST-511599, School of Computing Science, University of Newcastle, 2005.
 [3] L. Bettini, V. Bono, R. D. Nicola, G. Ferrari, D. Gorla, M. Loret, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri. The Klaim Project: Theory and Practice. In C. Priami, editor, *Global Computing: Programming Environments, Languages, Security and Analysis of Systems, LNCS 2874*, pages 88–150. Springer-Verlag, 2003.
 [4] F. Cristian. Exception Handling and Fault Tolerance of Software Faults. In M. Lyu, editor, *Software Fault Tolerance*, pages 81–107. Wiley, NY, 1995.
 [5] R. J. Dufresne, W. J. Gerace, W. J. Leonard, J. P. Mestre, and L. Wenk. Classtalk: A Classroom Communication System for Active Learning. *Journal of Computing in Higher Education*, 7:3–47, 1996.
 [6] G. Gay, M. Stefanone, M. Grace-Martin, and H. Hembrooke. The Effects of Wireless Computing in Collaborative Learning Environments. *International Journal of Human-Computer Interaction*, 13(2):257–276, 2001.
 [7] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
 [8] W. G. Griswold, P. Shanahan, S. W. Brown, R. Boyer, M. Ratto, R. B. Shapiro, and T. M. Truong. ActiveCampus - Experiments in Community-Oriented Ubiquitous Computing. *IEEE Computer*, 37(10):73–81, 2004. <http://activecampus.ucsd.edu/> [Last accessed: 1 Feb 2006].
 [9] A. Iliasov. Implementation of Cama Middleware. <http://sourceforge.net/projects/cama> [Last accessed: 1 Feb 2006].
 [10] A. Iliasov, L. Laibinis, A. Romanovsky, and E. Troubitsyna. Towards Formal Development of Mobile Location-based Systems. Presented at REFT 2005 Workshop on Rigorous Engineering of Fault-Tolerant Systems, Newcastle Upon Tyne, UK (<http://rodin.cs.ncl.ac.uk/events.htm>), June 2005.
 [11] A. Iliasov and A. Romanovsky. CAMA: Structured Coordination Space and Exception Propagation Mechanism for Mobile Agents. Presented at ECOOP 2005 Workshop on Exception Handling in Object Oriented Systems: Developing Systems that Handle Exceptions. July 25, 2005. Glasgow, UK, 2005.
 [12] A. Iliasov and A. Romanovsky. Exception Handling in Coordination-based Mobile Environments. In *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005)*, pages 341–350. IEEE Computer Society Press, 2005.
 [13] I. Merrick and A. Wood. Coordination with Scopes. In *Proceedings of the ACM Symposium on Applied Computing 2000*, pages 210–217, 2000.
 [14] A. Omicini and F. Zambonelli. Tuple Centres for the Coordination of Internet Agents. In *SAC '99: Proceedings of the 1999 ACM symposium on Applied computing*, pages 183–190, New York, NY, USA, 1999. ACM Press.
 [15] G. P. Picco, A. L. Murphy, and G.-C. Roman. Lime: Linda Meets Mobility. In *Proceedings of 21st Int. Conference on Software Engineering (ICSE'99)*, pages 368–377, 1999.
 [16] M. Ratto, R. B. Shapiro, T. M. Truong, and W. G. Griswold. The ActiveClass Project: Experiments in Encouraging Classroom Participation. In *Computer Support for Collaborative Learning 2003*, pages 477–486. Kluwer, 2003.
 [17] G.-C. Roman, C. Julien, and J. Payton. A Formal Treatment of Context-Awareness. In M. Wermelinger and T. Margaria, editors, *Fundamental Approaches to Software Engineering, 7th International Conference, FASE 2004, part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, LNCS 2984*, pages 12–36. Springer, 2004.
 [18] I. Satoh. MobileSpaces: A Framework for Building Adaptive Distributed Applications using a Hierarchical Mobile Agent System. In *Proceedings of the ICDCS 2000*, pages 161–168, 2000.
 [19] A. Tripathi and R. Miller. Exception Handling in Agent-oriented Systems. In *Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, pages 304–315. ACM Press, 2002.
 [20] P. van Roy, S. Haridi, P. Brand, G. Smalka, M. Mehl, and R. Scheidhauer. Mobile Objects in Distributed Oz. *ACM Transactions on Programming Languages and Systems*, 19(5):804–851, 1997.

This is an advance copy of an invited paper to be presented at the 19th IFIP World Computer Congress, 20-25 August 2006, Santiago, Chile. The presentation is on 22 August at the 1st IFIP International Conference on Biologically Inspired Cooperative Computing (BICC). The paper will be published in the *Proceedings of IFIP BICC 2006* by Springer Science and Business Media.

AN IMMUNE SYSTEM PARADIGM FOR THE ASSURANCE OF DEPENDABILITY OF COLLABORATIVE SELF-ORGANIZING SYSTEMS

Algirdas Avizienis

**Vytautas Magnus University, Kaunas, Lithuania
and**

**University of California, Los Angeles, USA
aviz@adm.vdu.lt**

Abstract

In collaborative self-organizing computing systems a complex task is performed by relatively simple autonomous agents that act without centralized control. Disruption of a task can be caused by agents that produce harmful outputs due to internal failures or due to maliciously introduced alterations of their functions. The probability of such harmful outputs is minimized by the application of a design principle called "the immune system paradigm" that provides individual agents with an all-hardware fault tolerance infrastructure. The paradigm and its application are described in this paper.

1. Dependability Issues of Collaborative Self-Organizing Systems

Self-organizing computing systems can be considered to be a class of distributed computing systems. To assure the dependability of conventional distributed systems, fault tolerance techniques are employed [1]. Individual elements of the distributed system are grouped into clusters, and consensus algorithms are implemented by members of the cluster [2], or mutual diagnosis is carried out within the cluster.

Self-organizing systems differ from conventional distributed systems in that their structure is dynamic [3]. Relatively simple autonomous agents act without central control in jointly carrying out a complex task. The dynamic nature of such systems makes the implementation of consensus or mutual diagnosis impractical, since constant membership of the clusters of agents cannot be assured as the system evolves. An agent that suffers an internal fault or external interference may fail and produce harmful outputs that disrupt the task being carried out by the collaborative system. Even more harmful can be maliciously introduced (by intrusion or by malicious software) alterations of the agent's function that lead to deliberately harmful outputs.

The biological analogy of the fault or interference that affects an agent is an infection that can lead to loss of the agent's functions and also to transmission of the infection to other agents that receive the harmful outputs, possibly causing an epidemic. The biologically inspired solution that I have proposed is the introduction within the agent of a fault tolerance mechanism, called the fault tolerance infrastructure (FTI), that is analogous to the immune system of a human being [4,5]. Every agent has its own FTI and therefore consensus algorithms are no longer necessary to protect the system.

2. A Design Principle: the Immune System Paradigm

My objective is to design the FTI for an autonomous agent that is part of a self-organizing system. I assume that the agent is composed of both hardware and software subsystems and communicates to other agents via wireless links. Then I will employ the following three analogies to derive a design principle called “the immune system paradigm”:

- (1) the human body is analogous to hardware,
- (2) consciousness is analogous to software,
- (3) the immune system of the body is analogous to the fault tolerance infrastructure FTI.

In the determination of the properties that the FTI must possess four fundamental attributes of the immune system are especially relevant [6]:

- (1) It is a part of the body that functions (i.e. detects and reacts to threats) continuously and autonomously, independently of consciousness.
- (2) Its elements (lymph nodes, other lymphoid organs, lymphocytes) are distributed throughout the body, serving all its organs.
- (3) It has its own communication links – the network of lymphatic vessels.
- (4) Its elements (cells, organs, and vessels) themselves are self-defended, redundant and in several cases diverse.

Now we can identify the properties that the FTI must have in order to justify the immune system analogy. The are as follows:

- (1a) The FTI consists of hardware and firmware elements only.
- (1b) The FTI is independent of (that is, it requires no support from) any software of the agent, but can communicate with it.
- (1c) The FTI supports (provides protected decision algorithms for) multichannel computing by the agent, including diverse hardware and software channels that provide design fault tolerance for the agent’s hardware and software.
- (2) The FTI is compatible with (i.e., protects) a wide range of the agent’s hardware components, including processors, memories, supporting chipsets, discs, power supplies, fans and various peripherals.
- (3) Elements of the FTI are distributed throughout the agent’s hardware and are interconnected by their own autonomous communication links.
- (4) The FTI is fully fault-tolerant itself and requires no external support. It is not susceptible to attacks by intrusion or malicious software and is not affected by natural or design faults of the agent’s hardware and software.
- (5) An additional essential requirement is that the FTI provides status outputs to those other agents with which it can communicate. The outputs indicate the state of the agent’s health: perfect or undergoing recovery action. Upon failure of the agent’s function the FTI shuts down all its outputs and issues a permanent status output indicating failure.

The above listed set of design requirements is called the immune system paradigm. It defines an FTI that can be considered to be the agent’s immune system that defends its “body” (i.e., hardware) against “infections” caused by internal faults, external interference, intrusions, and attacks by malicious software. The FTI also informs the other agents in its environment of its state of health. Such an FTI is generic, that is, it can serve a variety of agents. Furthermore it is transparent to the agent’s software, compatible with other defenses used by the agent, and fully self-protected by fault tolerance.

A different and independently devised analogy of the immune system is the “Artificial Immune System” (AIS) of S. Forrest and S. A. Hofmeyr [7]. Its origins are in computer security research, where the motivating objective was protection against illegal intrusions. The analogy of the body is a local-area broadcast network, and the AIS protects it by detecting connections that are not normally observed on the LAN. Immune responses are not included in the model of the AIS, while they are the essence of the FTI.

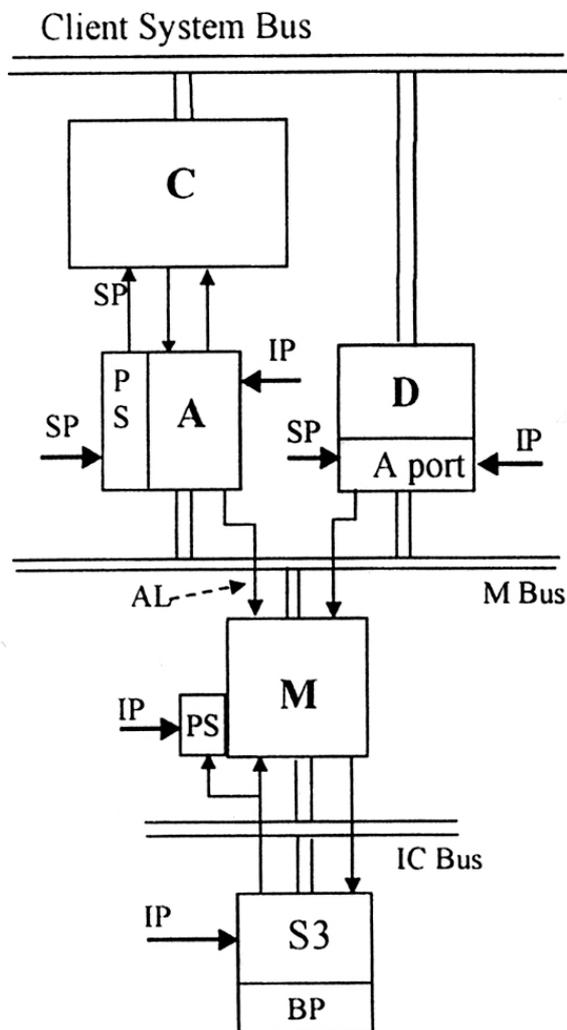
3. Architecture of the Fault Tolerance Infrastructure

The preceding sections have presented a general discussion of an FTI that serves as the analog of an immune system for the hardware of an agent of a self-organizing system. Such an FTI can be placed on a single hardware component, or it can be used to protect a board with several components, or an entire chassis [5]. To demonstrate that the FTI is a practically implementable and rather simple hardware structure, this and the next section describe an FTI design that was intended to protect a system composed of Intel P6 processors and associated chip sets and was first presented in [5].

The FTI is a system composed of four types of special-purpose controllers called “nodes”. The nodes are ASICs (Application-Specific Integrated Circuits) that are controlled by hard-wired sequencers or by read-only microcode. The basic structure of the FTI is shown in Figure 1. The figure does not show the redundant nodes needed for fault tolerance of the FTI itself. The C (Computing) node is a COTS processor or other hardware component of the agent being protected by the FTI. One A (Adapter) node is provided for each C node. All error signal outputs and recovery command inputs of the C node are connected to its A node. Within the FTI, all A nodes are connected to one M (Monitor) node via the M (Monitor) bus. Each A node also has a direct input (the A line) to the M node. The A nodes convey the C node error messages to the M node. They also receive recovery commands from the M node and issue them to C node inputs.

The A line serves to request M node attention for an incoming error message. The M node stores in ROM the responses to error signals from every type of C node and the sequences for its own recovery. It also stores system configuration and system time data and its own activity records. The M node is connected to the S3 (Startup, Shutdown, Survival) node. The functions of the S3 node are to control power-on and power-off sequences for the entire agent, to generate fault-tolerant clock signals and to provide non-volatile, radiation-hardened storage for system time and configuration. The S3 node has a backup power supply (e.g. a battery) and remains on at all times during the life of the FTI.

The D (Decision) node provides fault-tolerant comparison and voting services for the C nodes, including decision algorithms for N-version software executing on diverse processors (C-nodes). Fast response of the D node is assured by hardware implementation of the decision algorithms. The D node also keeps a log of disagreements in the decisions. The second function of the D node is to serve as a communication link between the software of the C nodes and the M node. C nodes may request configuration and M node activity data or send power control commands. The D node has a built-in A node (the A port) that links it to the M node. Another function of the FTI is to provide fault tolerant power management for the entire agent system, including individual power switches for every C node, as shown in Figure 1. Every node except the S3 has a power switch. The FTI has its own fault-tolerant power supply (IP).



- SP:** System Power
- IP:** Infrastructure Power
- BP:** Backup Power
- PS:** Power Switch
- C:** Computing Node
- A:** Adapter Node
- D:** Decision Node
- M:** Monitor Node
- S3:** Startup, Shutdown, Survival Node
- AL:** A-Line

Note: Redundant nodes are not shown

Figure 1. Basic Structure of the FTI

4. Fault Tolerance of the FTI

The partitioning of the FTI is motivated by the need to make it fault-tolerant. The A and D nodes are self-checking pairs, since high error detection coverage is essential, while spare C and D nodes can be provided for recovery under M node control. The M node must be continuously available, therefore triplication and voting (TMR) is needed, with spare M nodes added for longer life.

The S3 nodes manage M node replacement and also shut the agent down in the case of failure or global catastrophic events (temporary power loss, heavy radiation, etc.). They are protected by the use of two or more self-checking pairs with backup power. S3 nodes were separated from M nodes to make the node that must survive catastrophic events as small as possible. The S3 nodes also provide outputs to the agent's environment that indicate the health status of the agent: perfect, undergoing protective action or failed.

The all-hardware implementation of the FTI makes it safe from software bugs and external attacks. The one exception is the power management command from C to M nodes (via the D node) which could be used to shut the system down. Special protection is needed here. Hardware design faults in the FTI nodes could be handled by design diversity of self-checking pairs and of M nodes, although the logic of the nodes is very simple and their complete verification should be possible.

When interconnected, the FTI and the original autonomous agent form a computing system that is protected against most causes of system failure. An example system of this type is called DiSTARS: Diversifiable Self Testing And Repairing System and is discussed in detail in [5]. DiSTARS is the first example of an implementation of the immune system paradigm. Much detail of implementation of the FTI is presented in the U.S. patent application disclosure "Self-Testing and – Repairing Fault Tolerance Infrastructure for Computer Systems" by A. Avižienis, filed June 19, 2001.

5. In Conclusion: Some Challenges

The use of the FTI is likely to be affordable for most agents, since the A, M, D, and S3 nodes have a simple internal structure, as shown in [5] and the above mentioned disclosure. It is more interesting to consider that there are some truly challenging missions that can only be justified if their computing systems with the FTI have very high coverage with respect to design faults and to catastrophic transients due to radiation. Furthermore, extensive sparing and efficient power management can also be provided by the FTI. Given that the MTBF of contemporary processor and memory chips is approaching 1000 years, missions that can be contemplated include the 1000-day manned mission to Mars [8] with the dependability of a 10-hour flight of a commercial airliner. Another fascinating possibility is an unmanned very long life interstellar mission using a fault-tolerant relay chain of modest-cost DiSTARS type spacecraft [9]. Both missions are discussed in [5].

References

- [1] A.Avižienis, J.-C. Laprie, B. Randell, C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. On Dependable and Secure Computing*, 1(1):11-33, January-March 2004.
- [2] N.A. Lynch. Distributed algorithms. Morgan Kaufmann, 1996.
- [3] F. Heylighen, C. Gershenson, The meaning of self-organization in computers. *IEEE Intelligent Systems*, July/August 2003, pp. 72-75.
- [4] A. Avižienis. Toward systematic design of fault-tolerant systems. *Computer*, 30(4):51-58, April 1997.
- [5] A.Avižienis, A fault tolerance infrastructure for dependable computing with high performance COTS components. *Proc. of the Int. Conference on Dependable Systems and Networks (DSN 2000)*, New York, June 2000, pages 492-500.
- [6] G.J.V. Nossal. Life, death and the immune system. *Scientific American*, 269(33)52-62, September 1993.
- [7] S.A. Hofmeyr, S. Forrest. Immunity by design: An artificial immune system. *Proc. 1999 Genetic and Evolutionary Computation Conference*, pages 1289-1296. Morgan-Kaufmann, 1999.
- [8] Special report: sending astronauts to Mars. *Scientific American*, 282(3):40-63, March 2000.
- [9] A. Avižienis. The hundred year spacecraft. *Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware*, Pasadena, CA, July 1999, pp. 233-239.

Towards an Engineering Approach to Component Adaptation

Steffen Becker¹, Antonio Brogi², Ian Gorton³, Sven Overhage⁴,
Alexander Romanovsky⁵, and Massimo Tivoli⁶

¹ Software Engineering Group, University of Oldenburg,
OFFIS, Escherweg 2, 26121 Oldenburg, Germany
steffen.becker@informatik.uni-oldenburg.de

² Department of Computer Science, University of Pisa,
Largo B. Pontecorvo 3, 56127 Pisa, Italy
brogi@di.unipi.it

³ Empirical Software Engineering Group, National ICT Australia,
Bay 15 Locomotive Workshop, Australian Technology Park Eveleigh, NSW 1430 Australia
ian.gorton@nicta.com.au

⁴ Dept. of Software Engineering and Business Information Systems,
Augsburg University, Universitätsstraße 16, 86135 Augsburg, Germany
sven.overhage@wiwi.uni-augsburg.de

⁵ School of Computing Science, University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, United Kingdom
alexander.romanovsky@ncl.ac.uk

⁶ Dept. of Computer Science, University of L'Aquila,
Via Vetoio N.1, 67100 L'Aquila, Italy
tivoli@di.univaq.it

Abstract. Component adaptation needs to be taken into account when developing trustworthy systems, where the properties of component assemblies have to be reliably obtained from the properties of its constituent components. Thus, a more systematic approach to component adaptation is required when building trustworthy systems. In this paper, we illustrate how (design and architectural) patterns can be used to achieve component adaptation and thus serve as the basis for such an approach. The paper proposes an adaptation model which is built upon a classification of component mismatches, and identifies a number of patterns to be used for eliminating them. We conclude by outlining an engineering approach to component adaptation that relies on the use of patterns and provides additional support for the development of trustworthy component-based systems.

1 Introduction

In an ideal world, component-based systems are assembled from pre-produced components by simply plugging perfectly compatible components together, which jointly realize the desired functionality. In practice, however, it turns out that the constituent components often do not fit one another and adaptation has to be done to eliminate the resulting mismatches.

R.H. Reussner et al. (Eds.): Architecting Systems, LNCS 3938, pp. 193–215, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Mismatches between components, for example, always need to be addressed when integrating *legacy systems*. Thereby, the impossibility of modifying large client applications is a major reason for the need to employ some form of adaptation during the development of component-based systems. Besides that, the most structured way to deal with *component evolution* and upgrading, which is likely to result in new mismatches at the system level, arguably is by applying adaptation techniques. Finally, adaptation becomes a major task in the emerging area of *service-oriented computing*, where mismatches must be solved to ensure the correct interoperation among different Web services, which have been assembled according to a bottom-up strategy.

For these reasons, the adaptation of components has to be recognized as an unavoidable, crucial task in Component-Based Software Engineering (CBSE). Until now, however, only a number of isolated approaches to eliminate mismatches between components have been proposed. They introduce adapters, which are capable of mediating the interaction between components (e.g., to transform data between different formats or to ensure a failure-free coordination protocol). In [1, 2, 3, 4, 5, 6, 7, 8, 9], for instance, the authors show how to automatically derive adapters in order to reduce the set of system behaviors to a subset of safe (e.g., lock-free) ones. Other papers [9, 10, 11, 12, 13, 14] show how to plug a set of adapters into a system in order to augment the system behaviour by introducing more sophisticated interactions among components. The presented protocol transformations can be applied to ensure the overall system dependability, improve extra-functional characteristics, and properly deal with updates of the system architecture (e.g., insertion, replacement, or removal of components).

The approaches mentioned above only address some forms of component mismatch types, employ specific specification formalisms, and usually do not support any reasoning about the impact that component adaptation has on the extra-functional properties (e.g., reliability, performance, security) of the system. For these reasons, employing these approaches to adapt components in an ad hoc strategy typically is error prone, reduces the overall system quality, and thus increases the costs of system development. Above all, employing such an ad hoc strategy to component adaptation hinders the development of *trustworthy component-based systems*, since it is impossible to reliably deduce the properties of component assemblies from the properties of the constituent components and the created adapters.

To counter these problems, it is the objective of this paper to initiate the development of an *engineering approach* to component adaptation that provides developers with a systematic solution consisting of methods, best practices, and tools. As the basis for such an approach we suggest the usage of *adaptation patterns*, since they provide generic and systematic solutions to eliminate component mismatches. Before establishing the details of the proposed engineering approach, we start by clarifying important concepts (section 2). After introducing an initial taxonomy of component mismatches (section 3), we describe a generic process model for component adaptation and discuss relevant patterns that have emerged both in literature and in practice (section 4). To illustrate the employment of patterns to eliminate component mismatches, we additionally present some examples (section 5). After discussing related work we conclude by outlining some of the remaining challenges. They will have to be solved to establish

a fully-fledged engineering approach, capable of supporting the development of trustworthy component-based systems.

2 Component Adaptation: Coming to Terms

Component mismatches originate from contradicting assumptions about the context, in which interacting components should be used, and the real context, in which they are being deployed. These contradicting assumptions have been made by the developers of individual components and become obvious during the assembly of the system, when individual components are brought together. Component mismatches have been examined both from an architectural [15] and a reuse-oriented perspective [16].

From a reuse-oriented perspective there always is a tension between the goals of extending the functionality of a component on the one hand and keeping it reusable on the other hand. These are contradicting goals, since reuse typically requires simple, well-defined and well-understood functionality. Because of this reason, it is likely that a reused component will not exactly fit the required context. In the software reuse community, component mismatches are usually called *component incompatibilities*.

From a software architecture perspective, problems occur when components have different assumptions about normal and abnormal behaviour of other components or when a software architect makes decisions which contradict individual assumptions of the components and connectors [17]. Problems of this kind are called *architectural mismatches*. In our paper, we summarize the terms "component incompatibilities" and "architectural mismatches" as *component mismatches* to emphasize that they relate to the same problem.

Before we elaborate the proposed engineering approach to component adaptation, some terms have to be clarified as they are not used consistently in the domain of adaptation techniques.

Software adaptation is the sequence of steps performed whenever a software entity is changed in order to comply with requirements emerging from the environment in which the entity is deployed. Such changes can be performed at different stages during the life cycle. Therefore, we distinguish requirement adaptation, design-time adaptation, and run-time adaptation (see [18]):

- Requirement adaptation is used to react to changes during requirements engineering, especially when new requirements are emerging in the application domain.
- Design-time adaptation is applied during architectural design whenever an analysis of the system architecture indicates a mismatch between two constituent components.
- Run-time adaptation takes place when parts of the system offer different behaviour depending on the context the parts are running in. This kind of adaptation is therefore closely related to context-aware systems.

In the following, we restrict ourselves to design-time adaptation.

Software Component Adaptation is the sequence of the steps required to bridge a component mismatch. According to the common definition, components offer services to the environment, which are specified as provided interfaces [19, 20]. In addition,

components explicitly and completely express their context dependencies [19], i.e. their expectations on the environment. Context dependencies are stated in the form of required interfaces [19, 20]. Using the concept of provided and required interfaces, a component mismatch can be interpreted as a mismatch between properties of required and provided interfaces, which have to be connected (see figure 1). Consequently, identifying mismatches between components is equivalent to identifying mismatches between interfaces. A component mismatch thus occurs, when a component, which implements a provided interface, and a component, which uses a required interface, are not cooperating as intended by the designer of the system.

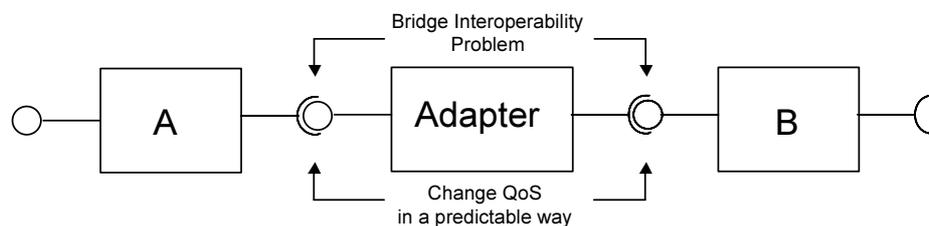


Fig. 1. A software component adapter and its QoS impact

Note that component mismatches explicitly refer to interoperability problems which have not been foreseen by the producer of one of the components. Many components offer so called *customization interfaces* to increase reusability. These interfaces allow changes to the behaviour of the component during assembly time by setting parameters. As they are foreseen by the component developer and thus planned in advance, we do not consider parameterization as adaptation. Therefore, in the following customization is disregarded.

In accordance with the term *adaptation* we define a *software component adapter* as a software entity especially constructed to overcome a component mismatch.

3 A Taxonomy of Component Mismatches

Although an efficient technique to adapt components is of crucial importance to facilitate CBSE, there currently exist only a few approaches to enumerate and classify different kinds of component mismatches [21]. Moreover, many of the existing approaches just broadly distinguish between *syntactic*, *semantic*, and *pragmatic mismatches* and put them into relation to various aspects of compatibility like *functionality*, *architecture*, and *quality* [22, 23]. In order to get a more detailed understanding of the problem domain, we start implementing the proposed engineering approach to component adaptation by introducing a *taxonomy of mismatches*. The introduced taxonomy enumerates different types of component mismatches which will be taken into consideration when we develop a pattern-based approach to adaptation later on.

In addition, the provided taxonomy summarizes the different types of component mismatches into categories and classifies them according to a hierarchy of *interface models* (see figure 2). Each of the distinguished interface models determines a (distinct)

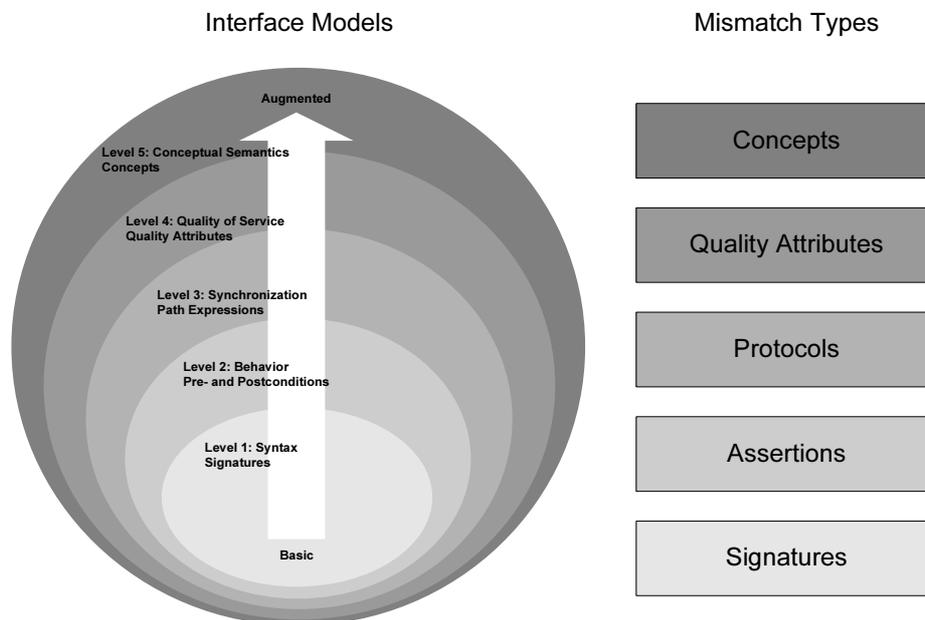


Fig. 2. A hierarchy of interface models (based on [24]), which orders interface properties according to their specification complexity, supports the identification and elimination of different types of component mismatches

set of properties which belongs to a component interface [24, 25]. Because component mismatches originate from mismatching properties of connected interfaces (the so-called provided and required interfaces, cf. section 2), the hierarchy of interface models underlying the interface descriptions simultaneously determines our ability to diagnose and eliminate a certain type of component mismatch.

The (classical) *syntax-based interface model*, which focuses on signatures as constituent elements of component interfaces, supports the identification and elimination of *signature mismatches*. By using such a syntax-based interface model, the following types of (adaptable¹) mismatches can be distinguished when connecting the required interface of a component "A" with a provided interface of a component "B" as shown in figure 1 [26, 27]:

- Naming of methods. Methods, which have been declared in the provided and required interface, realize the same functionality but have different names.
- Naming of parameters. Parameters of corresponding methods represent the same entity and have the same type but have been named differently in the provided and required interface.
- Naming of types. Corresponding (built-in or user-defined) types have been declared with different names.

¹ An adaptable mismatch can eventually be eliminated by adaptation.

- Structuring of complex types. The member lists of corresponding complex types (e.g. structures) declared both in the provided and required interface are permutations.
- Naming of exceptions. Exceptions thrown by corresponding methods have the same type, but have been declared with different names.
- Typing of methods. The method declared in the provided interface returns a type, which is a sub-type of the one that is returned by the method declared in the required interface.
- Typing of parameters. Parameters of methods declared in the provided interface have a type, which is a super-type of the one that belongs to corresponding parameters declared in the required interface.
- Typing of exceptions. Exceptions thrown by methods declared in the provided interface have a type, which is a sub-type of the one that belongs to corresponding exceptions declared in the required interface.
- Ordering of parameters. The parameter lists of corresponding methods declared both in the provided and required interface are permuted.
- Number of parameters. A method declared in the provided interface has fewer parameters or additional parameters with constant values compared to its corresponding method declared in the required interface.

Compared to this basic interface model, a *behavioral interface model* also contains *assertions* (i.e. pre- and postconditions) for the methods, which have been declared in the required and provided interfaces. With a behavioral interface model in place, it becomes principally conceivable to additionally search for (adaptable) mismatches between assertions when comparing provided and required interfaces. However, we chose *not* to consider the detection and adaptation of mismatching assertions as part of the proposed engineering approach, since they usually cannot be statically identified in an efficient manner [28, p. 578]. Instead, we refer to [29] for details about existing techniques, which can be applied to identify and adapt mismatching assertions, as well as their principal limitations.

By making use of an *interaction-based interface model*, which focuses on describing the interaction that takes place between connected components in the form of message calls, developers are able to diagnose and eliminate *protocol mismatches*. Provided that the interaction protocols belonging to the provided and the required interface are specified in a way that supports an efficient, i.e. statically computable, comparison, the following (adaptable) mismatches can be distinguished [2, 30]:

- Ordering of messages. The protocols belonging to the provided and required interface contain the same kinds of messages, but the message sequences are permuted.
- Surplus of messages. A component sends a message that is neither expected by the connected component nor necessary to fulfil the purpose of the interaction.
- Absence of messages. A component requires additional messages to fulfil the purpose of the interaction. The message content can be determined from outside.

Since it is generally possible to specify interaction protocols as pre- and postconditions [28, p. 981-982], we have to admit that introducing a behavioral interface model already would have been sufficient to cover the interaction aspect as well. Nevertheless, we

chose to view interaction protocols as a separate aspect that has to be distinguished from pre- and postconditions. This decision is mainly motivated by the problems that arise when trying to statically compare assertions. We have to admit, however, that our decision to view interaction protocols as a separate aspect only is profitable, if the specified interaction protocols can be statically compared in a more efficient way than assertions. To ensure a better comparability of protocol specifications, we eventually have to prefer notations of limited expressive power (e.g. finite state machines).

The *quality-based interface model* instead focuses on describing an aspect that has not been covered so far. It documents the Quality of Service (QoS) which is being provided by each of the interface methods by describing a set of quality attributes. The set of quality attributes that is to be described is determined by the underlying quality model, e.g. the ISO 9126 quality model [31, 32], which is one of the most popular. By making use of an interface model that is based on the ISO 9126 quality model, it is possible to detect and eliminate the following *quality attribute mismatches*:

- Security. The component requiring a service makes assumptions about the authentication, access, and integrity of messages that differ from the assumptions made by the component which provides the service.
- Persistency. The component requiring a service makes assumptions about the persistent storage of computed results that differ from the assumptions made by the component which provides the service.
- Transactions. The component requiring a service makes assumptions about the accompanying transactions that differ from the assumptions made by the component which provides the service.
- Reliability. The service required by component A needs to be more reliable than the one that is being provided by component B. Reliability is a trustworthiness attribute characterizing the continuity of the service, e.g. by measuring the meantime between failure, mean downtime, or availability [32, p. 23]. Typically reliability is achieved by employing fault tolerance means.
- Efficiency (Performance). The service required by component A needs to be more efficient than the one that is being provided by component B. The efficiency of a service is typically characterized by its usage of time and resources, e.g. the response time, throughput, memory consumption, or utilization of processing unit [32, pp. 42-50].

It is important to stress the fact that, with respect to adaptation, the quality aspect is a *cross-cutting concern*. This means, creating and inserting an adapter to eliminate one of the other component mismatches mentioned in this paper probably influences the quality properties, e.g. by delaying the response time of a service that now has to be invoked indirectly. In fact, the quality attributes distinguished above are even cross-cutting concerns among each other, which means that adapting one of the quality attributes is likely to influence the others.

A *conceptual interface model*, which describes the conceptual semantics of component interfaces as an ontology (i.e. a set of interrelated concepts), supports the identification and elimination of so-called *concept mismatches*. Thereby, concepts can principally characterize each of the elements contained in a syntactical interface model. Thus, they may refer to entities (such as parameters, type declarations etc.), functions (methods),

and processes (protocols). By making use of a concept model that consists of a term (denominator), an intension (definition), and an extension (corresponding real objects), the following *concept mismatches* can be principally distinguished [33, 34]:

- Synonyms. Two concepts, which characterize corresponding interface elements of a provided and required interface, are identical with respect to their definition, but have been used with different terms (e.g. customer and buyer).
- Sub- and Superordination. Two concepts, which characterize corresponding interface elements of a provided and required interface, are in a specialization or generalization relationship to each other.
- Homonyms. Two concepts, which characterize corresponding interface elements of a provided and required interface, are named with the same term but have different definitions (e.g. price as price including value-added tax and price as price without value-added tax).
- Equipollences. Two concepts, which characterize corresponding interface elements of a provided and required interface, have the same extension. However, they have different definitions which only share some common aspects (e.g. customer and debtor).

Both conceptual interface models, which make use of ontologies to describe the semantics of component-interfaces, as well as their usage for compatibility tests and adaptation are still under research. Consequently, there currently is little substantial support that can help in detecting and adapting concept mismatches (an overview of approaches can be found in [35, 36]). However, conceptual interface models are helpful in detecting and eliminating certain kinds of signature mismatches, like e.g. methods with identical functionality and different namings.

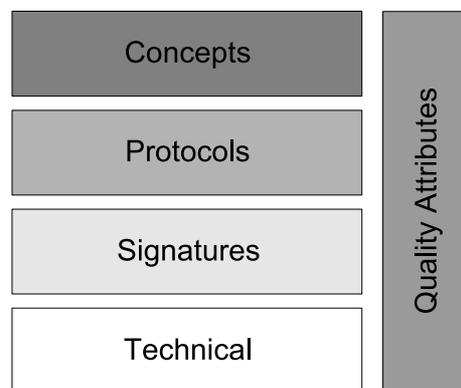


Fig. 3. The taxonomy contains five distinct classes of component mismatches

To complete our taxonomy of component mismatches, we finally introduce *technical mismatches* as additional component mismatch type. Technical mismatches between components occur, if two interacting components have been developed for different platforms (i.e. operating systems, frameworks etc.). Since technical dependencies of the

former kind usually are not described as interfaces and instead remain as implicit component properties, they have not been covered by the introduced taxonomy so far, which builds upon the hierarchy of interface models to classify component mismatches. They represent an important mismatch type, however, and have to be considered accordingly when developing an engineering approach to adaptation. Figure 3 shows the classification of component mismatches that results from the inclusion of technical mismatches. It shows extra-functional mismatches as cross-cutting concern, whereas the other concerns can be summarized as functional mismatches.

4 Relevant Patterns for Component Adaptation

Patterns - either on the component design or on the architectural level - have become popular since the Gang of Four [37] published their well-known book on design patterns. According to our classification there are a lot of possible component incompatibilities. Therefore, it is reasonable that there are several patterns for bridging those incompatibilities. As patterns are established and well known solutions to reoccurring problems we decide to utilize patterns for adaptation problems. Thus, in this section we highlight some of the relevant patterns - mainly taken from literature [37, 38, 39, 40].

Before we go into details, we focus on the basic structure of some of the patterns. Many patterns look similar or even identical at the design or source code level. This leads to the assumption that there are even more basic concepts used in the patterns than the patterns itself. For example, delegation is such a concept. Delegation takes place whenever a component wrapping another component uses the wrapped components service to fulfil its own service. For example, an adapter (see below) converting currencies from Euro to US Dollar. It first converts the input currency, then it delegates the call to the wrapped component using the right currency, and afterwards the currency is translated back again. The same idea is used also in, i.e., the Decorator pattern. Therefore, we try to identify in the following text these basic structures as well to build a taxonomy of the basic building blocks of the patterns introduced and also to capture the *basic technique* used in the patterns mentioned. An analysis of the basic techniques can also lead to a more engineering-based approach to adaptation in future work.

We classify the introduced patterns according to section 2 basically in adapters dealing primarily with functional aspects and extra-functional aspects respectively.

4.1 Functional Adaptation Patterns

This section gives an overview of the most often used patterns to bridge functional component incompatibilities. Most are well known to experienced developers and used quite frequently - even without the knowledge that a pattern has been used.

Adapter. The adapter or wrapper pattern is described in [37, p. 139]. The pattern directly corresponds to the definition given in section 2 as its main idea is to bridge between two different interfaces. The pattern is used in different flavours: a variant using inheritance and a second one based on delegation. The latter can be used in component-based development by using the concepts on the component instance level instead of the object instance level. The adapter pattern is very flexible as theoretically every interface can be transformed into every other interface. Thus, the range of adapters is infinite.

Decorator. The decorator pattern [37, p. 175] can be seen as a special class of adapters where the adapter's interface is a subtype of the adapted component. This enables the use of a decorated component instead of the undecorated. Additionally, it is possible to decorate a single component as often as necessary. As the adapted component has the same list of signatures as the original component, a decorator can only change or add functionality to the methods already offered by the original component. As the decorator is a special kind of adapter it also uses delegation as main technique.

Interceptor. Often the term interception is used when implementing aspect oriented programming (AOP) techniques. Interception is a technique which intercepts method calls and presents the call to some pre- or post-code for additional processing [39, p. 109]. It can be realized by the afore-mentioned decorator pattern but is often part of component runtime environments. For example, the J2EE container technology uses interception to add advanced functionality to components during deployment like container managed persistency or security. Basically, it also uses delegation but as said before often hidden in the runtime environments.

Wrapper Facade. The wrapper facade pattern is used to encapsulate a non-object oriented API using wrapper objects [39, p. 47]. Therefore, it can also be used to encapsulate services in a component-based framework. The basic idea is to encapsulate corresponding state and functions operating on this state in a single component. For example, consider a file system component encapsulating a file handle and the operations which can be performed on the respective file. Basic principles used in this pattern are delegation and the encapsulation of state.

Bridge. The bridge pattern is used to decouple an abstraction and its implementation [37, p. 151]. Thus, it is often used to define an abstract interface on a specific technology and its implementations deal with vendor specific implementations. Abstract GUI toolkits like Swing which can be used on top of different GUI frameworks can be seen as example. The basic technique here is the use of the subtype relation and polymorphism.

Microkernel. The Microkernel pattern uses a core component and drivers to build an external interface to emulate a specific environment [38, p. 171]. It can be used to simulate a complete target environment on a different technological platform. The pattern has been used for adaptation in writing emulation layers or virtual machines.

Mediator. The Mediator pattern is used to encapsulate how a given set of objects interact [37, p. 283]. A typical scenario in the context of adaptation is to use several components to provide a service, e.g., querying multiple database servers to return a single result set. The components can interact using the mediator's coordinating role. Often mediation is used simultaneously with the adapter pattern to transform data passed to or from the service in formats being expected by the respective interfaces. With a focus on data transformations the pattern is often also called *Coordinator* pattern [40, p. 111].

4.2 Extra-Functional Adaptation Patterns

The extra-functional patterns selected here are often used to increase a single or several quality attributes of the components being adapted. We give examples of properties that are often addressed by the patterns in the respective paragraphs.

Proxy. A Proxy is put in front of a component to control certain aspects of the access to it [37, p. 207]. Security issues like access control, encryption, or authentication are often added to components by respective Proxys. Additionally, it can be used to implement caching strategies [40, p. 83] or patterns for lazy acquisition of resources [40, p. 38] to increment response times. The basic technique used in this pattern is delegation.

Component Replication. The component replication pattern is derived from the object replication pattern [41, p. 99]. The idea is to distribute multiple copies of the same component to several distinct computation units to increase response time and throughput. Additionally, you might get an increased reliability in the case the controller coordinating the replicated components is not the point of failure. The basic technique in this pattern is based on copying the state of a component.

Process Pair. The process pair pattern runs each component twice so that one component can watch the other and restart it in case of a failure [41, p. 133]. The pattern is used to increase the availability of components in high availability scenarios, e.g., whenever safety is an important aspect of the system design. The basic principle of this pattern is based on timeouts.

Retransmission. Retransmission is used when a service call might vanish or fail [41, p. 187]. In case the failure lasts for a short period of time, e.g., a network transmission failure, a retransmission results in successful execution. Thus the pattern increases the reliability of the system - especially when unreliable transactions are involved. The pattern is based on timeouts combined with a respective retry strategy.

Caching. The cache pattern keeps data retrieved from a slower memory in a faster memory area to allow fast access if an object is accessed twice [40, p. 83]. Therefore, the pattern is used to increase response time and throughput. The benefits are acquired by accepting a larger memory footprint. The basic technique of the pattern uses memory buffers to increase performance.

Pessimistic Offline Lock. The pessimistic offline lock is a pattern used to control concurrent access to components or resources controlled by components [42, p. 426]. The lock is used to ensure that solely one single thread of execution is able to access the protected resource. Hence, the lock ensures certain safety criteria on the cost of performance as concurrent threads have to wait before they can execute. The basic principle used in the pattern is based on blocking the control flow using the process scheduler.

Unit of Work. The unit of work pattern is used to collect a set of sub-transactions in memory until all parts are complete and then commits the whole transaction by accessing the database only a short time [42, p. 184]. Like the cache pattern there is a trade-off

	Technical	Signatures	Protocols	Concepts	Quality Attributes
Adapter		✓	✓	✓	✓
Decorator				✓	
Interceptor				✓	✓
Wrapper Facade		✓		✓	
Bridge	✓				
Microkernel	✓				
Mediator			✓		
Proxy					✓
Replication					✓
Process Pair					✓
Retransmission					✓
Caching					✓
Pessimistic Lock					✓
Unit of Work					✓

Fig. 4. A classification of Patterns and Mismatches

between memory consumption and efficiency. As in the caching pattern the basic idea is to use a memory buffer.

4.3 Classification of Patterns

The collection of patterns does not claim to be complete, there are more patterns which we could look at. We introduced it to show that there are a lot of patterns which can be used to adapt components - mostly in a way which is not producing hand written glue code. In the table in figure 4 we show which patterns can be used to solve problems of the introduced mismatch classes.

5 Using Patterns to Eliminate Component Mismatches

After introducing a set of patterns in the previous section, we will now discuss how to use the patterns in a software engineering process. First, we will introduce a generic process which is supposed to serve as a guideline for adaptation. We will illustrate its usage by giving an example of a functional and an extra-functional adaptation. In particular, we will show an application of the Adapter/Wrapper pattern and of the Caching pattern.

The process of adapting components in order to construct trustworthy component assemblies using software engineering consists of the following steps:

1. *Detect mismatches:* First the mismatch between the required and provided interface has to be detected. As stated above, this directly depends on the specifications available, i.e., if no protocol specification is available then we can not detect protocol mismatches.

2. *Select measures to overcome the mismatch*: Second, we select from a set of established methods the one which is known to solve the specific mismatch. Note, that this choice also depends on the specifications available as some patterns can only be distinguished by examining subtle differences in the target setting (as already mentioned in section 4). This can sometimes require semantic information which is hard to analyze automatically. It is therefore necessary in many cases to leave the final choice to the developer. Nevertheless, it is possible to filter unsuitable patterns out in advance.
3. *Configure the measure*: Often the method or pattern selected can be fine-tuned as patterns are described as *abstract* solutions to problems. Thereby, we can for instance utilize the specifications and query the developer for additional input. If the specification is complete the solution to the mismatch problem is analyzed.
4. *Predict the impact*: After determining the solution of the problem we predict the impact of the solution on our setting. This is common in other engineering disciplines.
5. *Implement and test the solution*: If the prediction indicates that the mismatch is fixed, the solution is implemented, either by systematic construction or by using generative technologies.

5.1 Adapting Functional Mismatches with the Adapter Pattern

This section shows how functional adaptation can be implemented by utilizing the Adapter/Wrapper pattern [37, p. 139]. As shown in the table in figure 4, this pattern might be used to repair syntax, protocol and semantics mismatches.

The Adapter pattern (also known as Wrapper pattern) maps the interface of a component onto another interface expected by its clients. The Adapter lets components work together that could not otherwise because of incompatible interfaces. The participants in the “schema” of this pattern are: (i) the existing component interface that needs to be adapted, usually denoted as *Adaptee*; (ii) *Target* is the interface required by a client component and it is not compatible to *Adaptee*; (iii) *Client* denotes any client whose required interface is compatible to *Target* and (iv) *Adapter*, which is the component responsible for making *Adaptee* compatible to *Target*.

Here, we discuss an example of a possible application of the Adapter pattern seen as a means to overcome only protocol mismatches. Let us suppose that we want to assemble a component-based cooling water pipe management system that collects and correlates data about the amount of water that flows in different water pipes. The water pipes are placed in two different zones, denoted by *P* and *S*, and they transport water that has to be used to cool industrial machinery. The system we want to assemble is a client-server one. The zones *P* and *S* have to be monitored by a server component denoted as *Server*. *Server* allows the access to a collection of data related to the water pipes it monitors. It provides an interface denoted as *IServer*. Since some of the water pipes do not include a *Programmable Logic Controller* (PLC) system, *Server* cannot always automatically obtain the data related to the water that flows in those water pipes. Therefore, *IServer* exports the methods *PCheckOut* and *SCheckOut* to get an exclusive access to the data collection related to the water which flows in the pipes. This allows a client to: (i) read the data automatically stored by the server and (ii) manually update

the report related to the water which flows in the pipes that are not monitored by a PLC. Correspondingly, *IServer* exports also the methods *PCheckIn* and *SCheckIn* to both publish the updates made on the data collection and release the access gained to it. We want to assemble the discussed client-server system formed by the following selected components: *Server* and one client denoted as *Client*. The interface required by *Client* is compatible to *IServer* at level of both signature and semantics.

According to step 1 of the presented process, we need to be able to detect possible protocol mismatches. These days, we can utilize UML2 *Sequence Diagrams* and *Interaction Overview Diagrams* (i.e., the UML2 *Interaction Diagrams* suite) to extend the IDL specification of a component interface for including information related to the component interaction protocol. UML2 sequence diagrams are for describing a single execution scenario of a component or a system; UML2 interaction overview diagrams can be used to compose all the specified component/system execution scenarios into execution flows to indicate how each scenario fit together different ones during the overall execution of the component/system (see Figure 5).

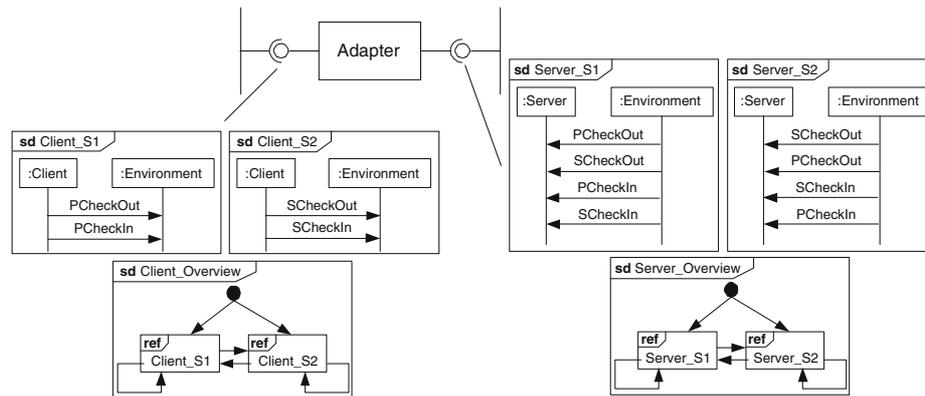


Fig. 5. An example of UML2 Interaction Diagrams specification to detect protocol mismatches

From the UML2 specification shown in Figure 5, it is possible to check automatically that the interaction protocols expected by *Server* and *Client* mismatch. That is, the selected server component forces its clients to always access to the data collections related to the zone *P* and *S* subsequently and in any possible order, before releasing the access gained for both of them. Instead, the selected client component gains the access and releases it for the data collections related to the zone *P* and *S* separately. This protocol mismatch leads to a deadlock.

According to step 2 of our proposed engineering approach to component adaptation, we have to choose the right type of measure to solve the problem. We decide to deploy an Adapter/Wrapper component to force a “check-out” of the data collection related to the zone *S* (*P*) after the client has performed a *PCheckOut* (*SCheckOut*) method call. The release of the gained access is handled analogously. In doing so, the interaction protocol of *Client* is enhanced in order to match the interaction protocol of *Server* (i.e., to avoid the deadlock). This adaptation strategy can be automatically derived by a tool

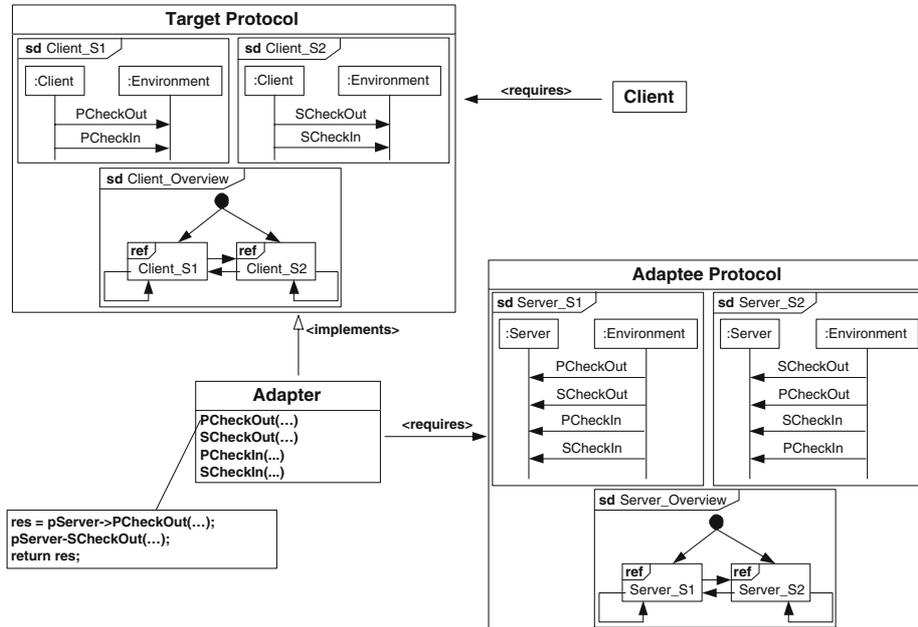


Fig. 6. Overall structure of the Adapter/Wrapper pattern to avoid protocol mismatches

that - by exploiting the UML2 XMI - is able to take in input an XML representation of the UML2 interaction diagrams specification of *Server* and *Client*. This tool might elaborate - in some way - this specification and produce the adaptation strategy that must be implemented by the Adapter. A similar approach can be found in [14].

In the third step of our process we have to customize the pattern to our needs (i.e., protocol adaptation purposes) and choose the right variant of it. We plan to implement the pattern according to Figure 6 depicting the overall structure of our realization.

The following are the participants to the Adapter pattern applied to bridge protocol mismatches: (i) **Target Protocol** which is the protocol required by a client component (i.e., the interaction protocol of *Client*); (ii) **Client** which is a component whose protocol is compatible to the Target Protocol (i.e., *Client*); (iii) the **Adapter** which is the component responsible for making an existing protocol compatible to the Target Protocol; and (iv) the **Adaptee Protocol** which is the existing protocol (i.e., the interaction protocol of *Server*). In the figure we also show a portion of the code implementing the method *PCheckOut* as provided by the Adapter component. *SCheckOut*, *PCheckIn* and *SCheckIn* are implemented analogously. This code reflects the adaptation strategy discussed above.

In the next step, in order to make it an engineering process, we predict the impact of the deployed Adapter in terms of checking whether the detected protocol mismatch has been solved or not. To be able to do so, we do not need any further information beyond the UML2 Interaction Diagrams specification of *Client* and *Server* and the underlined structure of the Adapter component. In fact, from this kind of specification, it is possible to automatically derive a process algebra notation e.g., FSP notation [43], of

the interaction behavior of *Client*, *Server* and of the Adapter component. FSP notation might be a useful formalism to check automatically if the insertion of the Adapter in the system will avoid the detected protocol mismatch. In the literature, there are more functional analysis tools that support FSP as input language.

One of these tools is LTSA (*Labeled Transition System Analyser*) [43]. LTSA is a plugin-based verification tool for concurrent systems. It checks automatically that the specification of a concurrent system satisfies required properties of its behavior such as deadlock-freeness. Thus, by integrating our process with such tools we can predict whether the detected protocol mismatch will be solved by the Adapter component or not. Moreover - since the Adapter also changes extra-functional properties of the system, e.g., by slowing down accesses because of the injected method calls - we should predict the impact of the Adapter on the performance of method calls. In the next subsection it is very clearly explained how to predict it by using an usage profile of an adapted service. Here, we simply note that performance of method calls should decrease but very little because the Adapter adds only a lightweight extra-level of indirectness.

In the final step, the adapter is built by exploiting the information contained in its pattern description. Depending on the complexity of the Adapter, this can be done either mechanically by a tool or by the developers. Once the Adapter is deployed, tests that validate both the results of the prediction and the adapter correctness are performed.

5.2 Adapting Extra-Functional Mismatches with the Caching Pattern

In the following we show how extra-functional adaptation can be achieved by employing the Caching pattern [40, p. 83]. A cache is used if a service needs some kind of resource whose acquisition is time consuming and the resource is not expected to change frequently but to be used often. The idea is to acquire the resource and to put it in the cache afterwards. The resource can be retrieved faster from the cache than re-acquiring it again. This is often done by utilizing additional memory to store the resource for faster retrieval. Hence, a trade-off is established between retrieval time and memory consumption. If the resource is needed again, it is retrieved from the cache. Often a validation check is performed in advance to test whether the cached resource is still up to date. Additionally, if the resource is altered by its usage we have to ensure consistency with the non-cached original object. This can be done by either storing it at its original location directly when the resource is altered (write-through-strategy). The other option is that the resource gets stored as soon as it gets evicted from the cache.

According to step 1 of the presented process, we need to be able to detect the mismatching response times. These days, we can utilize QML [44] specifications of the respective interfaces for this task. For example, let's assume an average response time of 3000ms is needed and an average response time of 6000ms is provided for service under investigation (see figure 7).

Additionally, we know that the required service processes requests to a static database. Therefore, we can consider the database table rows in the above stated sense. The database is not updated frequently, so caching the database query results will improve the average response time. Note, that we also need to know that the service fulfills these prerequisites of the cache pattern. It is to automatically determine if the prerequisites

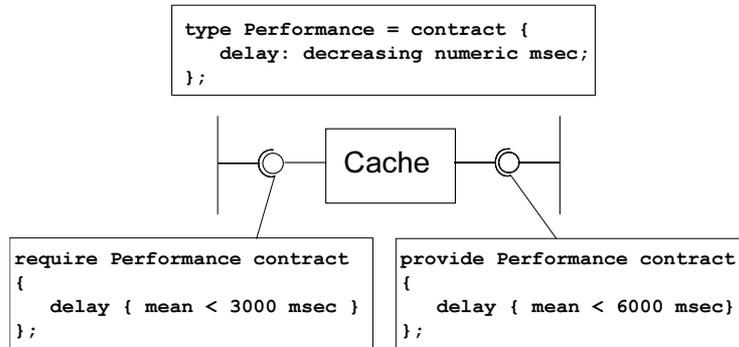


Fig. 7. An example QML specification to detect a QoS mismatch

are fulfilled as service specifications often state nothing about the resource usage of the specified service.

Second, we have to choose the right type of measure to solve the problem. We decide to deploy a cache to speed up an encapsulated resource access in the component being used. In doing so, the response time is decreased and the components can interoperate as desired.

In the third step we have to customize the pattern to our needs and choose the right variant of the pattern. Referring to the description in [40] we have to

- Select resources: The database query results
- Decide on an eviction strategy: Here we can choose between well-known types like least recently used (LRU), first in - first out (FIFO), and so on.
- Ensure consistency: We need a consistency manager whose task is to invalidate cache entries as soon as the master copy is changed. In the given database scenario it makes no sense to omit that part.
- Determine cache size: How much memory the cache is going to use. Most likely this is specified in number of cacheable resource units.

We plan to implement the pattern according to the following figure depicting the static structure of our realization (see figure 8).

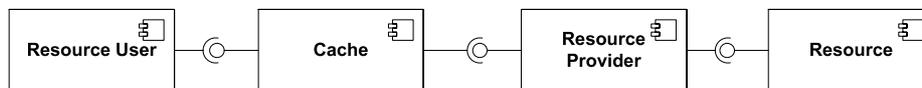


Fig. 8. The cache pattern implemented with components

In the next step, in order to make it an engineering process, we predict the impact of the deployed cache. To be able to do so, the usage profile of the adapted service is needed, as the performance of a cache depends on it. The usage profile information needed in this context, is the (estimated) frequency and type of requests. Together with the decisions taken in the previous step a specialized prediction model for the cache

impact can be applied and the result is compared to the requirements. This step is not well researched so that today we often neglect the step and trust on the experience of the deployer. Future work might come up with more prediction models to enable the engineering process as depicted here. To continue, let us assume, that the result is 2500ms and thus, the mismatch is resolved.

In the final step the adapter is finally constructed or generated by using the instructions given in the respective pattern description. Once the adapter is deployed, we perform tests to ensure that the predictions have been right and that everything works as expected.

6 Related Work

Even though Component-Based Software Engineering was first introduced in 1968 [45], developing systematic approaches to adaptation of components in order to resolve interoperability problems is still a field of active research. Many papers are based on the work done by Yellin and Strom [2, 46], who introduced an algorithm for the (semi-) automatic generation of adapters using protocol information and an external adapter specification. Bracciali et al. propose the use of some kind of process calculus to enhance this process and generate adapters using PROLOG [47].

Schmidt and Reussner present adapters for merging and splitting interface protocols and for a certain class of protocol interoperability problems [30]. Besides adapter generation, Reussner's parametrized contracts also represent a mechanism for automated component adaptation [48]. Additionally, Kent et al. [49] propose a mechanism for the handling of concurrent access to a software component not built for such environments.

Vanderperren et al. have developed a tool called PaCoSuite for the visual assembly of components and adapters. The tool is capable of (semi-)automatic adapter generation using signature and protocol information [50]. Gschwind uses a repository of adapters to dynamically select a fitting adapter [51]. Min et al. present an approach called Smart Connectors which allows the construction of adapters based on the provided and required interface of the components to connect [27].

Passerone, de Alfaro and Henzinger developed a game-theoretical approach to find out whether incompatible component interfaces can be made compatible by inserting a converter between them which satisfies specified requirements [4]. This approach is able to automatically synthesize the converter. Their approach can only be applied to a restricted class of component mismatches (protocols and interaction). In fact, they are only able to restrict the system's behavior to a subset of desired ones and, for example, they are not able to augment the system's behavior to introduce more sophisticated interactions among components.

In [10], Garlan et al. have shown how to use formalized protocol transformations to augment the interaction behavior of a set of components. The key result was the formalization of a useful set of interaction protocol enhancements. Each enhancement is obtained by composing wrappers. This approach characterizes wrappers as modular protocol transformations. The basic idea is to use wrappers to introduce more sophisticated interactions among components. The goal is to alter the behavior of a component with respect to the other components in the system, without actually modifying the

component or the infrastructure itself. While this approach deals with the problem of enhancing component interactions, it does not provide a support for wrapper generation.

A common terminology for the Quality of Service prediction of systems which are being assembled from components is proposed in [52]. A concrete methodology for predicting extra-functional properties of .NET assemblies is presented in [53]. None of these approaches, however, provides a specialized method for including adapters in their predictions. Engineering Quality of Service guarantees in the context of distributed systems is the main topic of [54].

An overview on adaptation mechanisms including non-automated approaches can be found in [55] (such as delegation, wrappers [37], superimposition [56], metaprogramming (e.g., [57])). Both works also contain a general discussion of requirements for component adaptation mechanisms. Not all of these approaches can be seen as adapters as defined in this paper. But some of the concepts presented can be implemented in adapters as shown here.

7 Conclusions and Future Directions

This paper introduces an engineering approach to software component adaptation. We define adaptation in terms of dealing with component mismatches, introduce the concept of component mismatch, and present a taxonomy to distinguish different types of component mismatches. Afterwards, we discuss a selection of adaptation patterns that can be used to eliminate the different mismatch types. The main contribution of the paper is a presentation of how these patterns can be used during the component adaptation process. The presented approach is demonstrated by both a functional and an extra-functional adaptation example.

Future research is directed towards exploring additional interface description languages which enable the efficient checking of the introduced mismatch types. On the basis of the available specification data, algorithms have to be developed to statically check for the identified component mismatch types during a compatibility test. Further on, existing prediction methods, which are based on the available component data, have to be improved to include adaptation and its impact on extra-functional system properties. In doing so, measures have to be developed that assess the impact on the extra-functional properties of systems when applying specific patterns to identified adaptation problems.

The application of generative techniques or concepts of Model-Driven Architecture (MDA) to construct the appropriate adapters is another strand of ongoing work. In this context, dependable composition of adapters and generation of adapters from the specification of the integrated system and the components are emerging areas of research. Finally, to achieve a fully-fledged engineering approach to component adaptation, further effort will be required to develop suitable tools that are capable of supporting the selection of pattern(s) which can be applied to solve specific mismatch types (viz., step 2 of the process proposed in Sect. 5).

Acknowledgments

The authors would like to thank Viktoria Firus, Gerhard Goos, and Raffaella Mirandola for their valuable and inspiring input during our break-out session at Dagstuhl, which

preceded this paper. Steffen Becker is funded by the German Science Foundation in the DFG-Palladio project. Alexander Romanovsky is supported by the IST FP6 Project on Rigorous Open Development Environment for Complex Systems (RODIN).

References

1. Balemi, S., Hoffmann, G.J., Gyugyi, P., Wong-Toi, H., Franklin, G.F.: Supervisory Control of a Rapid Thermal Multiprocessor. *IEEE Transactions on Automatic Control* **38** (1993) 1040–1059
2. Yellin, D., Strom, R.: Protocol Specifications and Component Adaptors. *ACM Transactions on Programming Languages and Systems* **19** (1997) 292–333
3. de Alfaro, L., Henzinger, T.A.: Interface Automata. In Gruhn, V., ed.: *Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT Symposium on the Foundation of Software Engineering (ESEC/FSE-01)*. Volume 26, 5 of *ACM SIGSOFT Software Engineering Notes.*, New York, ACM Press (2001) 109–120
4. Passerone, R., de Alfaro, L., Henzinger, T., Sangiovanni-Vincentelli, A.L.: Convertibility Verification and Converter Synthesis: Two Faces of the Same Coin. In: *Proceedings of the International Conference on Computer Aided Design (ICCAD'02)*. (2002)
5. Giannakopoulou, D., Pasareanu, C.S., Barringer, H.: Assumption Generation for Software Component Verification. In IEEE, ed.: *17th IEEE International Conference on Automated Software Engineering (ASE 2002)*, 23-27 September 2002, Edinburgh, Scotland, UK, Los Alamitos, CA, IEEE Computer Society (2002) 3–12
6. Inverardi, P., Tivoli, M.: Software Architecture for Correct Components Assembly. In Bernardo, M., Inverardi, P., eds.: *Formal Methods for Software Architectures, Third International School on Formal Methods for the Design of Computer, Communication and Software Systems: Software Architectures, SFM 2003*, Bertinoro, Italy, September 22-27, 2003, *Advanced Lectures*. Volume 2804 of *Lecture Notes in Computer Science.*, Berlin, Heidelberg, Springer (2003) 92–121
7. Inverardi, P., Tivoli, M.: Failure-Free Connector Synthesis for Correct Components Assembly. In: *Proceedings of Specification and Verification of Component-Based Systems (SAVCBS'03)*. (2003)
8. Tivoli, M., Inverardi, P., Presutti, V., Forghieri, A., Sebastianis, M.: Correct Components Assembly for a Product Data Management Cooperative System. In Crnkovic, I., Stafford, J.A., Schmidt, H.W., Wallnau, K.C., eds.: *Component-Based Software Engineering, 7th International Symposium, CBSE 2004*, Edinburgh, UK, May 24-25, 2004, *Proceedings*. Volume 3054 of *Lecture Notes in Computer Science.*, Berlin, Heidelberg, Springer (2004) 84–99
9. Brogi, A., Canal, C., Pimentel, E.: Behavioural Types and Component Adaptation. In Ratray, C., Maharaj, S., Shankland, C., eds.: *Algebraic Methodology and Software Technology, 10th International Conference, AMAST 2004*, Stirling, Scotland, UK, July 12-16, 2004, *Proceedings*. Volume 3116 of *Lecture Notes in Computer Science.*, Berlin, Heidelberg, Springer (2004) 42–56
10. Spitznagel, B., Garlan, D.: A Compositional Formalization of Connector Wrappers. In IEEE, ed.: *Proceedings of the 25th International Conference on Software Engineering*, May 3-10, 2003, Portland, Oregon, USA, Los Alamitos, CA, IEEE Computer Society (2003) 374–384
11. Tivoli, M., Garlan, D.: Coordinator Synthesis for Reliability Enhancement in Component-Based Systems. Technical report, Carnegie Mellon University (2004)
12. Autili, M., Inverardi, P., Tivoli, M., Garlan, D.: Synthesis of 'Correct' Adaptors for Protocol Enhancement in Component-Based Systems. In: *Proceedings of Specification and Verification of Component-Based Systems (SAVCBS'04)*. (2004)

13. Autili, M., Inverardi, P., Tivoli, M.: Automatic Adaptor Synthesis for Protocol Transformation. In: Proceedings of the First International Workshop on Coordination and Adaptation Techniques for Software Entities (WCAT'04). (2004)
14. Tivoli, M., Autili, M.: SYNTHESIS: A Tool for Synthesizing 'Correct' and Protocol-Enhanced Adaptors. To appear on L'Objet journal, <http://www.di.univaq.it/tivoli/LastSynthesis.pdf> (2005)
15. Garlan, D., Allan, R., Ockerbloom, J.: Architectural Mismatch: Why Reuse Is So Hard. *IEEE Software* **12** (1995) 17–26
16. Mili, H., Mili, F., Mili, A.: Reusing Software: Issues and Research Directions. *IEEE Transactions on Software Engineering* **21** (1995) 528–561
17. de Lemos, R., Gacek, C., Romanovsky, A.: Architectural Mismatch Tolerance. In de Lemos, R., Gacek, C., Romanovsky, A., eds.: *Architecting Dependable Systems*. Volume 2677 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, Springer (2003) 175–194
18. Canal, C., Murillo, J.M., Poizat, P.: Coordination and Adaptation Techniques for Software Entities. In Malenfant, J., Østvold, B.M., eds.: *Object-Oriented Technology: ECOOP 2004 Workshop Reader, ECOOP 2004 Workshops*, Oslo, Norway, June 14-18, 2004, Final Reports. Volume 3344 of *Lecture Notes in Computer Science*, Springer (2005) 133–147
19. Szyperski, C., Gruntz, D., Murer, S.: *Component Software: Beyond Object-Oriented Programming*, 2 edn. ACM Press and Addison-Wesley, New York, NY (2002)
20. D'Souza, D.F., Wills, A.C.: *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, Reading, MA, USA (1999)
21. Becker, S., Overhage, S., Reussner, R.: Classifying Software Component Interoperability Errors to Support Component Adaption. In Crnkovic, I., Stafford, J.A., Schmidt, H.W., Wallnau, K.C., eds.: *Component-Based Software Engineering, 7th International Symposium, CBSE 2004*, Edinburgh, UK, May 24-25, 2004, Proceedings. Volume 3054 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, Springer (2004) 68–83
22. Yakimovich, D., Travassos, G., Basili, V.: A classification of software components incompatibilities for COTS integration. Technical report, Software Engineering Laboratory Workshop, NASA/Goddard Space Flight Center, Greenbelt, Maryland (1999)
23. Overhage, S., Thomas, P.: WS-Specification: Specifying Web Services Using UDDI Improvements. In Chaudhri, A.B., Jeckle, M., Rahm, E., Unland, R., eds.: *Web, Web Services, and Database Systems. NODe 2002 Web- and Database-Related Workshops*. Volume 2593 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, Springer (2003) 100–118
24. Beugnard, A., Jezequel, J.M., Plouzeau, N., Watkins, D.: Making Components Contract Aware. *IEEE Computer* **32** (1999) 38–45
25. Overhage, S.: UnSCom: A Standardized Framework for the Specification of Software Components. In Weske, M., Liggesmeyer, P., eds.: *Object-Oriented and Internet-Based Technologies, 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World, NODe 2004*, Proceedings. Volume 3263 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, Springer (2004) 169–184
26. Zaremski, A.M., Wing, J.M.: Signature Matching: A Tool for Using Software Libraries. *ACM Transactions on Software Engineering and Methodology* **4** (1995) 146–170
27. Min, H.G., Choi, S.W., Kim, S.D.: Using Smart Connectors to Resolve Partial Matching Problems in COTS Component Acquisition. In Crnkovic, I., Stafford, J.A., Schmidt, H.W., Wallnau, K.C., eds.: *Component-Based Software Engineering, 7th International Symposium, CBSE 2004*, Edinburgh, UK, May 24-25, 2004, Proceedings. Volume 3054 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany (2004) 40–47
28. Meyer, B.: *Object-Oriented Software Construction*. 2. edn. Prentice Hall, Englewood Cliffs, NJ (1997)

29. Zaremski, A.M., Wing, J.M.: Specification Matching of Software Components. *ACM Transactions on Software Engineering and Methodology* **6** (1997) 333–369
30. Schmidt, H.W., Reussner, R.H.: Generating Adapters for Concurrent Component Protocol Synchronisation. In: *Proceedings of the Fifth IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems*. (2002)
31. ISO/IEC: Software Engineering - Product Quality - Quality Model. ISO Standard 9126-1, International Organization for Standardization (2001)
32. ISO/IEC: Software Engineering - Product Quality - External Metrics. ISO Standard 9126-2, International Organization for Standardization (2003)
33. Horwich, P.: Wittgenstein and Kripke on the Nature of Meaning. *Mind and Language* **5** (1990) 105–121
34. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic Matchmaking of Web Services Capabilities. In Horrocks, I., Hendler, J., eds.: *First International Semantic Web Conference on The Semantic Web*. Volume 2342 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, Springer (2002) 333–347
35. Noy, N.F.: Tools for Mapping and Merging Ontologies. In Staab, S., Studer, R., eds.: *Handbook on Ontologies*. Springer, Berlin, Heidelberg (2004) 365–384
36. Noy, N.F.: Semantic Integration: A Survey Of Ontology-Based Approaches. *SIGMOD Record* **33** (2004) 65–70
37. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, USA (1995)
38. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: *Pattern-Oriented Software Architecture – A System of Patterns*. Wiley & Sons, New York, NY, USA (1996)
39. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: *Pattern-Oriented Software Architecture – Volume 2 – Patterns for Concurrent and Networked Objects*. Wiley & Sons, New York, NY, USA (2000)
40. Kircher, M., Jain, P.: *Pattern-Oriented Software Architecture: Patterns for Distributed Services and Components*. John Wiley and Sons Ltd (2004)
41. Grand, M.: *Java Enterprise Design Patterns: Patterns in Java (Patterns in Java)*. John Wiley & Sons (2002)
42. Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., Stafford, R.: *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional (2002)
43. Magee, J., Kramer, J.: *Concurrency: State Models and Java Programs*. John Wiley and Sons (1999)
44. Frølund, S., Koistinen, J.: *Quality-of-Service Specification in Distributed Object Systems*. Technical Report HPL-98-159, Hewlett Packard, Software Technology Laboratory (1998)
45. McIlroy, M.D.: “Mass Produced” Software Components. In Naur, P., Randell, B., eds.: *Software Engineering*, Brussels, Scientific Affairs Division, NATO (1969) 138–155 Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968.
46. Yellin, D., Strom, R.: Interfaces, Protocols and the Semiautomatic Construction of Software Adapters. In: *Proceedings of the 9th ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA-94)*. Volume 29, 10 of *ACM Sigplan Notices*. (1994) 176–190
47. Bracciali, A., Brogi, A., Canal, C.: A formal approach to component adaptation. *Journal of Systems and Software* **74** (2005) 45–54
48. Reussner, R.H.: Automatic Component Protocol Adaptation with the CoCoNut Tool Suite. *Future Generation Computer Systems* **19** (2003) 627–639
49. Kent, S.D., Ho-Stuart, C., Roe, P.: Negotiable Interfaces for Components. In Reussner, R.H., Poernomo, I.H., Grundy, J.C., eds.: *Proceedings of the Fourth Australasian Workshop on Software and Systems Architectures*, Melbourne, Australia, DSTC (2002)

50. Vanderperren, W., Wydaeghe, B.: Towards a New Component Composition Process. In: Proceedings of ECBS 2001 Int Conf, Washington, USA. (2001) 322 – 331
51. Gschwind, T.: Adaptation and Composition Techniques for Component-Based Software Engineering. PhD thesis, Technische Universität Wien (2002)
52. Hissam, S.A., Moreno, G.A., Stafford, J.A., Wallnau, K.C.: Packaging Predictable Assembly. In Bishop, J.M., ed.: Component Deployment, IFIP/ACM Working Conference, CD 2002, Berlin, Germany, June 20-21, 2002, Proceedings. Volume 2370 of Lecture Notes in Computer Science., Springer (2002) 108–124
53. Dumitrascu, N., Murphy, S., Murphy, L.: A Methodology for Predicting the Performance of Component-Based Applications. In Weck, W., Bosch, J., Szyperski, C., eds.: Proceedings of the Eighth International Workshop on Component-Oriented Programming (WCOP'03). (2003)
54. Aagedal, J.Ø.: Quality of Service Support in Development of Distributed Systems. PhD thesis, University of Oslo (2001)
55. Bosch, J.: Design and Use of Software Architectures – Adopting and evolving a product-line approach. Addison-Wesley, Reading, MA, USA (2000)
56. Bosch, J.: Composition through Superimposition. In Weck, W., Bosch, J., Szyperski, C., eds.: Proceedings of the First International Workshop on Component-Oriented Programming (WCOP'96), Turku Centre for Computer Science (1996)
57. Kiczales, G.: Aspect-oriented programming. *ACM Computing Surveys* **28** (1996) 154–154

Context-Aware Exception Handling in Mobile Agent Systems: The MoCA Case

Karla Damasceno¹

Nelio Cacho²

Alessandro Garcia²

Alexander Romanovsky³

Carlos Lucena¹

¹Computer Science Department, Pontifical Catholic University of Rio de Janeiro – PUC-Rio, Brazil

²Computing Department, Lancaster University, UK

³Computer Science School, University of Newcastle upon Tyne, UK

{karla,lucena}@inf.puc-rio.br, {n.cacho,a.garcia}@lancaster.ac.uk
alexander.romanovsky@newcastle.ac.uk

ABSTRACT

Handling erroneous conditions in context-aware mobile agent systems is challenging due to their intrinsic characteristics: openness, lack of structuring, mobility, asynchrony and increased unpredictability. Even though several context-aware middleware systems support now the development of mobile agent-based applications, they rarely provide explicit and adequate features for context-aware exception handling. This paper reports our experience in implementing error handling strategies in some prototype context-aware collaborative applications built with the MoCA (Mobile Collaboration Architecture) system. MoCA is a publish-subscribe middleware supporting the development of collaborative mobile applications by incorporating explicit services to empower software agents with context-awareness. We propose a novel context-aware exception handling mechanism and discuss some lessons learned during its integration in the MoCA infrastructure.

Keywords

Exception handling, mobile agents, mobile computing, pervasive computing, context-awareness, middleware, fault tolerance.

1. INTRODUCTION

There is a growing popularity of pervasive agent-based applications that allow mobile users to seamlessly exploit the computing resources and collaboration opportunities while moving across distinct physical regions. Typically mobile collaborative applications need to be made context aware to allow autonomous adaptation of the agent functionalities. In particular, they need to deal with frequent variations in the system execution contexts, such as fluctuating network bandwidth, temperature changes, decreasing battery power, changes in location or device capabilities, degree of proximity to other users, and so forth. However, the development of robust context-aware mobile systems is not a trivial task due to their intrinsic characteristics of openness, “unstructureness”, asynchrony, and increased unpredictability [6, 22].

These system features seems to indicate that the handling of exceptional situations in mobile applications is more challenging, which in turn makes it impossible the direct application of conventional exception handling mechanisms [20, 21]. First, error propagation needs to be context aware since it needs to take into consideration the dynamic system boundaries and changing collaborative agents. Second, both the execution of error recovery

activities and determination of exception handling strategies often need to be selected according to user contexts. Third, the characterization of an exception itself may depend on the context, i.e. a system state may be considered an erroneous condition in a given context, but it may be not in others.

Several middleware systems [6,14,23] are nowadays available to support the construction of mobile agent-based applications. Their underlying architecture rely on different coordination techniques, such as tuplespaces [6], publish-subscribe mechanisms [14], and computational reflection [23]. However, such the middleware systems rarely provide explicit support for *context-aware exception handling*. Often the existing solutions (e.g. [17,22,24]) are too general and not specific for the characteristics of the coordination technique used. Typically they are not scaleable because they do not support clear system structuring using exception handling contexts. Our analysis shows that understanding the interplay between context awareness and exception handling in mobile agent systems is still an open issue. As a result, in order to deal with the complexity of context-aware exceptions, application programmers need to directly rely on existing middleware mechanisms, such as interest subscriptions or regular tuple propagation. The situation is complicated even further when they need to express exceptional control flows in the presence of mobility.

We have implemented error handling features in several prototype context-aware collaborative applications built with the MoCA (Mobile Collaboration Architecture) system [14]. MoCA is a publish-subscribe middleware that supports the development of collaborative mobile applications by incorporating explicit services empowering software agents with context-awareness. This paper presents the lessons learned while developing exception handling in MoCA applications. We have identified a number of exception handling issues that are neither satisfied by the regular use of the exception mechanisms of programming languages nor addressed by conventional mechanisms of the existing context-aware middleware systems, such as MoCA.

The main contributions of this paper are as follows. First, we present a case study helping us to identify the requirements for the context-aware exception handling mechanism. The system is a typical ambient intelligence (AmI) application developed with the MoCA middleware. Secondly, using these requirements we formulate a proposal for a context-aware exception handling model. Thirdly, we describe a prototype implementation of the model in the MoCA middleware; it consists of an extension of the client and server APIs and new middleware services, such as

management of *exceptional contexts*, context-sensitive error propagation and execution of context-aware exception handlers. We also analyse the difficulties in using a typical publish-subscribe infrastructure for supporting context-aware exception handling.

The plan of the paper is as follows. Section 2 presents the basic concepts of context awareness, surveys context-aware middleware styles and introduces the fundamental exception handling concepts. Section 3 describes the case study in which we have identified challenging exception handling issues for the development of robust context-aware agent applications. Section 4 discusses an implementation of the proposed mechanism in MoCA. Section 5 overviews the related work. Section 6 concludes the paper by discussing directions of future work.

2. BACKGROUND

This section discusses the background of our work. Section 2.1 introduces the terminology and a categorization of the context-aware middleware. Section 2.2 overviews the MoCA system. Section 2.3 introduces the exception handling concepts used in this paper.

2.1 Context and Context-Aware Middleware

The concepts of context and context-aware systems have been defined in a number of ways (e.g. [2, 3, 4]). According to Dey and Abowd [1], context is any information that can be used to characterize the situation or an entity. A system is context-aware if it uses context to provide relevant information and/or services to the user. Thus, one entity can be represented by an agent or a person with a mobile device and the context-aware system can provide information about location, identity, time and activity for these entities. Before the context can be used it is necessary to acquire data from sensors, conduct context recognition and some other tasks [5]. These tasks are usually implemented by context-aware middleware, which hides the heterogeneity and distributed nature of devices processing the contextual information. In general, three types of architectural styles are used to implement context-aware middleware systems: (i) tuplespace-based architectures [9], (ii) reflective architectures [10,11] and (iii) publish/subscribe architectures [7].

Tuplespace is a form of distributed shared memory spread across all participant processes and/or hosts. Processes using this model communicate by generating *tuples* and *anti-tuples* which are submitted to the tuple space [9]. Tuples are typed data structures (e.g., objects in C++ and Java), each tuple is formed from a collection of typed data fields and represents a cohesive piece of information. In a tuplespace-based system, all inter-process communications are exclusively conducted using the tuple space and any process using a tuple space has the ability to access all the tuples it contains, dynamically insert new tuples, find matches for nondestructive anti-tuples and remove tuples by generating matching destructive anti-tuples [9]. Cama (Context-Aware Mobile Agents) [6] is an example of the tuplespace middleware. The four basic CAMA abstractions are location, scope, agent, and role. A location is a container for scopes. A scope provides a coordination space within which compatible agents can interact. In this framework, devices can move from location to location. Each location runs a host computer supporting wireless connectivity. This computer keeps and controls the local tuple space to be accessed from the devices connected locally. This

tuple space is the only media supporting communication between these devices.

Reflective middleware [10, 11] exploits mechanisms of computational reflection [12] to implement mobility and context-awareness services. Reflection is used to monitor the middleware internal (re)configuration [13]. Reflective middleware system is divided in two levels: *base level* and *meta level*. The base level represents the middleware and the application core. The meta level contains the building blocks responsible for supporting reflection. These two levels are connected through a meta-object protocol (MOP) to ensure that modifications at the meta level are reflected into the corresponding modifications at the base level. Thus, modifications at the core should be reflected at the meta-level. The elements of the base level and of the meta level are respectively represented by base-level objects and meta-level objects. For example reflection is explored in CARISMA [23] to enhance the construction of adaptive and context-aware mobile applications. The middleware provides software engineers with primitives to describe how context changes should be handled using policies. The reflective middleware is in charge of maintaining a valid representation of the execution context by directly interacting with the underlying network operating system. Applications may require some services to be delivered in different ways (using different policies) when requested in the different context.

Publish/Subscribe (pub/sub) architectures rely on an asynchronous messaging paradigm that allows loose coupling between publishers and subscribers. Publishers are the agents that send information to a central component, while subscribers express their interest in receiving messages. Broker [7] or Dispatcher [8] is the central component of a pub/sub system, and is responsible for recording all subscriptions, matching publications against all subscriptions, and notifying the corresponding subscribers. The following session describes MoCA, a context-aware publish/subscribe middleware which has been used in our first experiment to incorporate exception handling strategies in context-aware mobile agent systems. Such an architecture was selected because of the growing number of context-aware middleware systems based on the publish-subscribe model [7,8,14].

2.2 MoCA: Mobile Collaboration Architecture

MoCa [14] is a middleware system supporting development and execution of the context-aware collaborative applications which work with mobile users. Figure 1 shows the three elements that compose the MoCa application: a server, a proxy, and clients. The first two are executed on the nodes of the wired network, while the clients run on mobile devices. A proxy intermediates all communication between the application server and one or more of its clients on mobile hosts. The server and the client of a collaborative application are implemented using the MoCA APIs, which hide from the application developer most of the details concerning the use of the services provided by the architecture. The *ProxyFramework* white-box framework is used for developing and customizing the proxies according to the specific needs of the application [14]. It allows adaptation to be triggered by the context-change events.

The internal MoCA infrastructure is shown in Figure 2. To support context-aware applications, MoCA supplies three

services: Context Information Service (CIS), Symbolic Region Manager (SRM) and Location Inference Service (LIS). The CIS component receives and processes state information sent by the clients. It also receives notification requests for from the application Proxies, and generates and delivers events to a proxy whenever a change in a client's state is of interest to this proxy. To provide transparency, CIS takes decisions on behalf of the publish/subscribe mechanism; which is implemented using built-in mechanisms that cater for the basic functionalities rather than deal with the high levels of heterogeneity and dynamicity intrinsic to mobile environments, such as the problem of late delivery [7].

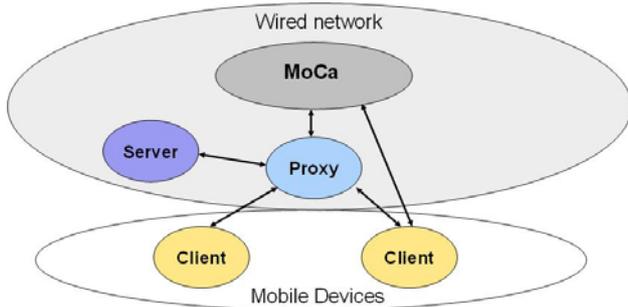


Figure 1. MoCA application

SRM provides an interface to define and request information about hierarchies of symbolic regions, which are names assigned to well-defined physical regions (i.e. rooms, halls, buildings) that may be of interest to location-aware applications [14]. Based on SRM information, LIS infers the approximate location of a mobile device from the raw context information collected by CIS of this device. It does this by comparing the current pattern of radio frequency (RF) signals with the signal patterns previously measured at the pre-defined Reference Points of the physical region. Therefore, to make any inference, the LIS database has to be populated with RF signal probes (device pointing in several directions) at each reference point, and with the inference parameters that are chosen according to the specific characteristics of the region.

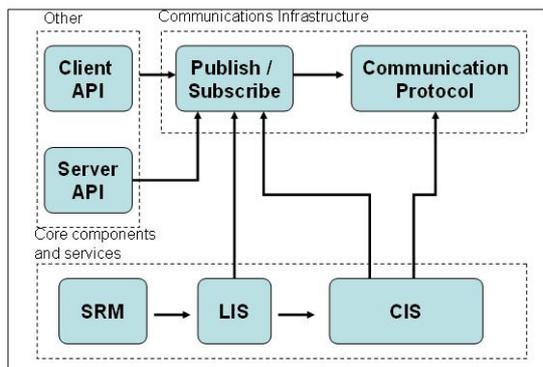


Figure 2. MoCA internal infrastructure

The communication infrastructure consists of the pub/sub mechanism and a communication protocol. The former supplies the basic functionality to the CIS, once the context recognition is done by the definition of subscriptions that specify a set of features to activate a specific user-defined context. The communication protocol currently works with an 802.11 wireless network based on the IP protocol stack, but the architecture could

be extended to accommodate a cellular data network protocol, such as GPRS.

2.3 Exception Handling

Agent activity, as the activity of any software component, can be divided into two parts [25]: *normal activity* and *exceptional activity*. The normal activity implements the agent's normal services while the exceptional activity provides measures that cope with *exceptions*. Each agent (and other system components) should have *exception handlers*, which constitute its exceptional activity. Handlers are attached to a particular region of the normal code which is called *protected region* or *handling scope*. Whenever an agent cannot handle an exception it raises, the exception will be signaled and *propagated* to other handling scopes defined in the higher-level components of the system. After the exception is handled, the system returns to its normal activity.

Developers of dependable systems often refer to errors as exceptions because they manifest themselves rarely during the agent's normal activity. Exceptions can be classified into two types [21]: (i) *user-defined*, and (ii) *pre-defined*. The user-defined exceptions are defined and detected at the application level. The predefined exceptions are declared implicitly and are associated with the erroneous conditions detected by the run-time support, the middleware or hardware .

Exception handling mechanisms [20,21] developed for many high-level programming languages allow software developers to define exceptions and to structure the exceptional activity of software component. An exception handling mechanism introduces the specific way in of *exception propagation* and of changing the normal control flow to the *exceptional control flow* when an exception is raised. It is also responsible for supporting different exceptional flow strategies and search for the appropriate handlers after an exception occurs. Exception mechanisms are either built as an inherent part of the language with its own syntax, or as a feature of the middleware systems coping with the intricacies of the different application domains and architecture styles.

3. Context-Aware Exception Handling in Mobile Agent Systems

This section describes a typical context-aware agent-based application, for which we have implemented a prototype system with the MoCA architecture, and identified a number of difficulties in incorporating error handling. Section 3.1 describes the case study, while Section 3.2 presents the identified requirements for a mechanism smoothly supporting context-aware exception handling.

3.1 AmI: a Case Study

The case study is an ambient intelligence (AmI) [16] application, which is composed of numerous sensors, devices and control units interconnected to effectively form a machine [15]. A wide range of sensors and controllers could be utilized, such as: fire alarm, energy control, heating control, ventilation control, climate, surveillance, lightning, power, and automatic door and window. Figure 3 depicts an AmI scenario where each office contains sensors and output devices, which are monitored and controlled locally by software agents. All these agents are connected

together via a network, forming a decentralized architecture that enables building-wide collaboration.

Each piece of equipment has an associated device controlling its activation. All users have a smartcard that operates as a mobile device supplying the current position and employee ID. Immediately after entered the office, the system needs to identify the user preferences and starts the procedures for dealing with the temperature, ventilation, illumination, and climate adaptation for the specific user preferences. In order to achieve the system robustness, a number of environmental, hardware, and software-related exceptions that need to be effectively handled. Such exceptional circumstances include fire or excessive number of users in a given building region, or the occurrence of problems in the diverse primary and/or secondary mobile heating systems distributed over the building, or even in the central heating system. The handling of such exceptional conditions depend on the combination of changing contextual information, such as the location and type of the heating systems, the physical regions where the different system administrators are, and so on.

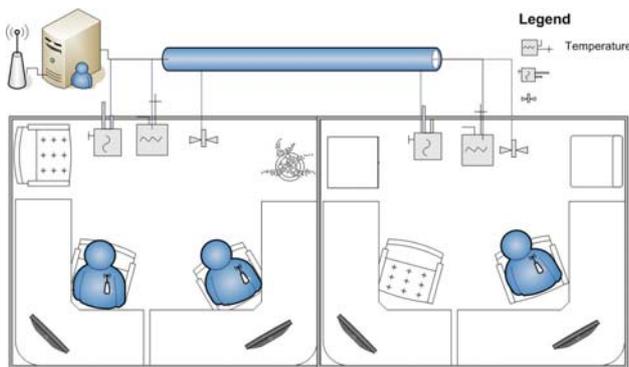


Figure 3. Plant: A floor of a typical building structured into offices. All sensors are wired to a common field-bus network.

In the following, we discuss problems relative to the incorporation and implementation of error handling scenarios in such a context-aware mobile agent-based application. First, we explain the problems found in the context of our case study. Second, we explain why they cannot be addressed while using the underlying mechanisms of the MoCA architecture. The shortcomings here vary from exception declaration to exception handlers and error propagation issues.

3.2 Specification of “Exceptional Contexts”

During design of the AmI application we have identified a number of user-defined “exceptional contexts” that depend on a multitude of contextual information and also on user preferences, which in turn are typically application-specific. For us, exceptional contexts mean one or more conditions associated with the context types, which together denote an environmental, hardware, or software fault. For example, the exceptional contexts can be characterized by the situations when the temperature of an office or public room in the building occasionally exceeds the maximum limit according to user preferences, which can indicate a serious problem in the heating system (not detected by the associated controlling system). Handling of such situations requires an exceptional control flow different from the normal one, consisting of regular notification-based reactions. The seriousness of this context requires propagation of such

exceptional context information to the proper administrators, which also may vary depending on their physical location. It may also require involvement of several people.

The specification of contextual conditions of interest in publish-subscribe systems, such as MoCA, requires explicit subscriptions based on regular expressions. The subscription is usually carried out by the code in the devices or proxy servers, which will be receiving notifications when those contextual conditions are matched according to the changing circumstances. However, the specification of an exceptional context situation inherently has a different semantics and, as such, needs to encompass different elements in its specification, including the handling scope, alternative “default handlers”, types of contextual information which *should* and *should not* be propagated together with the exception occurrence, and so on. This is why in MoCA normal contextual subscriptions need to be different from the exceptional subscriptions.

3.3 Lack of Exception Handling Scoping

There are several situations in the AmI case study (Section 3.1) when handling exceptions requires several software agents and users to be involved depending on the physical regions and other types of contextual information. For example, as discussed in Section 3.2, the proper handling of some exceptional conditions in the mobile heaters requires exceptions to be propagated to a set of devices belonging to the staff responsible for heater maintenance. However, the propagation needs to be context sensitive in the sense it should take into account the suitable maintainers for the specific heater type that is closest to the region where the faulty heater is located. The contextual exception needs to be systematically propagated to broader scopes until the appropriate handlers are found. Moreover, if some fire exception is detected, it needs to be propagated to all the building regions and group of mobile users. Hence the physical regions or a group of devices (such as, those ones with the maintenance people) are examples of contextual handling scopes that should be supported by the underlying middleware. In this way, the proper exception handlers could be activated in all the relevant devices according to different user preferences. However, the MoCA middleware does not support such scopes for context-aware error handling, which hinders the modularity of the system on the presence of exceptional contexts.

3.4 Need for Context-Aware Handlers

There are also some cases where the selection of proper exception handlers depends on the contextual conditions associated with devices involved in the coordinated error handling. For the same exception, we need to create handlers tailored to different contextual conditions, and make sure that they are correctly executed. For instance, we need to associate contextual information about the heater physical location to the handlers dealing with the faulty heaters. Some handlers can be only selected if the mobile heater is in the context of a specific department. Again, we have to implement such a control of context-aware handlers as part of the application since there is no MoCA facility for that purpose.

3.5 Awareness of Unforeseen Exceptions

In an open mobile system, like the AmI study (Section 3.1), we could not wait that all the devices, in which software agents were developed by different designers, would be able to foresee all the

exceptional contexts. In the AmI case, for example, the presence of fire in the building may not have been foreseen by all the designers of the software agents running in the mobile devices located in the different building regions. As a result, there was a need for exploiting the mobile collaboration infrastructure when an exceptional context is detected by one of the peers. Depending on the exception severity, it should be notified to other mobile devices even when they have not registered interest in that specific exceptional context. In other words, the contextual exception should be proactively raised in other mobile collaborative agents and/or mobile devices pertaining to the same region. Thus robust context-aware mobile systems require more intelligent, proactive exception handling due to their features of openness, asynchrony, and increased unpredictability. The problem is that conventional coordination models (Section 2.1) such as tuplespace-based and publish-subscribe architectures (e.g. MoCA), require the explicit subscription of interest from the collaborative agents.

4. Exception Handling in MoCA

This section presents our context-aware exception handling model and its MoCA implementation, which deal with the problems discussed in Sections 3.2 – 3.2.4. Our current approach basically supports the notion of *exceptional context* (Section 3.2) and *different levels of handling scopes* (Section 3.3) to treat the limitations of conventional APIs and mechanisms provided by existing context-aware middleware systems (Section 2.1).

Exceptional contexts. The goal of exceptional contexts is to facilitate the definition of exceptional situations in applications that have a great number of devices and sensors that collect information for a specific purpose. An exceptional context corresponds to undesirable or dangerous set of conditions pertaining to different contexts. They can be associated with one specific user, application’s agents, or mobile devices.

Scope nesting: protected device(s), regions, or groups. In order to support a modular context-aware approach for error propagation, exceptions can be caught by scopes at four different levels: a device, a group (of devices), a proxy server, and a region. In our MoCA implementation, the central MoCA server, where the CIS and LIS services (Section 2.2) are located, is also treated as an exception handling scope. To illustrate these types of handling scopes, Figure 4 depicts how scopes of different type covers different elements of our AmI case study (Section 3.1). Device and server scopes comprise basic operational units of the context-aware system, which allows the exception handler functionality to be encapsulated into the scope of its own unit (device or server).

Group-based scopes. Group scope encompasses a set of devices which are defined by the application to support mobile cooperative handling of an exception amongst the device’s agents pertaining to that group. This kind of scope is not directly related to spatial relationship, and it makes it possible to insert or remove elements from the scope according to the application necessity. Thus software agents can autonomously join and leave a group. For example, the agents acting on behalf of heat maintainers (Section 3.1) can form a specific group, as when heating-related exceptions are raised all of them may be notified.

Region-based scopes. Differently from the three first ones, a region scope has a more dynamic behavior to identify t-he devices

that is part of the scope. In our implementation, this scope is strongly related to the MoCa LIS service (Section 2.2) as it provides a reference mechanism that allows a device be aware of its neighbors. In addition, it is possible to control when a device enters and goes out from one region scope. A device movement characterizes the scope change; in other words, whenever a device moves from one a physical region X to Y, it automatically moves from the region-based handling scope X to Y. Hence this movement encompasses the context-sensitive change of the exceptional conditions that the device can handle.

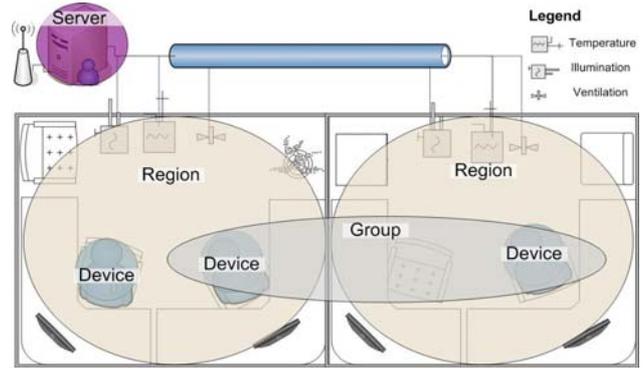


Figure 4. Different scope levels: Device, Region, Group, Server

Region hierarchy. Moreover, region scope can be organized in a hierarchy fashion in which is possible to define that, for example, *Office 1* is part of *Offices*, and subsequently, *Offices* is part of *Computing Dep.* that is also part of *University* region. This hierarchy allows a specific handler or exception for a specific region or sub-region to be defined. To support hierarchy arrangement, users should define the relationship between LIS symbolic regions and SRM hierarchy tree (Section 2.2). This can be done by the MoCa API.

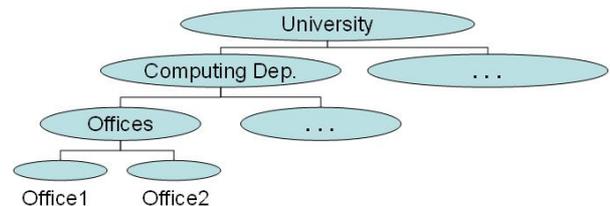


Figure 5. Scope hierarchy definition

Regular contextual subscription. Figure 6 shows the server code responsible for defining a user preference for a specific office. Due to the user movement, it is necessary to subscribe a listener in the LIS service to notify whenever a new device enters in the office. When this occurs, server gets user preferences and starts the appropriate services to suit the user requirements. To avoid long connections between server and devices, once each device can take long time to finish the process, servers break the connection after invoking the start method.

An exceptional context example. As discussed in Section 3, a device’s agent suddenly detects that the temperature is exceeding the maximum limit, and then, it infers that there is a fault in the mobile heating system. In this situation, the device should throw and propagate an exception to inform the other mobile users and

the heater support about the problem. This step is done by the code shown in Figure 7. The *UnableToHeat* exception is created and thrown by the heating device.

```

RegionListener listen = new RegionListen();
Lis_service.subscribe(UserDevice, listen);
...
private class RegionListen
    implements RegionListener {
    public void onDeviceEntered(String regionID,
                               String deviceID)
        HashMap userpref = getUserPref(deviceId,
                                         regionID);
        Heat.start(userpref.get("Temperature"));
        LightCtrl.start(userpref.get("Light"));
        . . .
    }
}

```

Figure 6. User preference definition.

Context-aware error propagation. It also specifies that it needs to be propagated to a set of scopes. To define the sequence of exception propagation, the user should use the *propagateTo* method. This method receives as parameter the scope reference, the sequence number, and also the condition constant. The scope reference determines for which scope the exceptional context will be propagated, the sequence number, and in which order. Scope constant defines the propagation policy in which the error propagation will occur; it determines when the exception needs to be propagated: whether when none (NONE value), one (ONE value) or all handlers (ALL value) were found and successfully executed to deal with this exception. For instance, in Figure 7, the *UnableToHeat* will firstly propagate to the region and group. If none of them handles this exception, it is delivered to the server scope.

Contextual information propagation and context-aware selection of handlers. In addition, the *UnableToHeat* exceptional context can carry information related to the exceptional occurrence. This may include the operation status of the thermostat, the operation status of the tip over the safety mechanism, the heater type, and also the heater brand. This contextual information is carried with the propagated exception to allow the exception mechanism to select an appropriate context-aware handler for a specific exception. Note that, after defining the exception content, this exception is thrown in the last line of the Figure 7.

```

UnableToHeat unaheat = new UnableToHeat();
Unaheat.propagateTo((RegionScope.getInstance(),1,
Scope.NONE);
Unaheat.propagateTo((RegionScope.getInstance("Heat
Maintainer"),1,Scope.NONE);
Unaheat.propagateTo((ServerScope.getInstance("Main
Server"),2);
Unaheat.getContext().
    setStringProperty("Thermostat","noanswer");
Unaheat.getContext().
    setStringProperty("tipoversafety","noanswer");
Unaheat.getContext().
    setStringProperty("HeaterType","Electric");
Unaheat.getContext().
    setStringProperty("Brand","HeaterCompanyA");
EHMMechanism.throw(unaheat);

```

Figure 7. Device code: throwing exception.

Context-sensitive handlers. In order to handle the *UnableHeat* exception, four handlers are defined in different scopes. The first one deals with the university maintenance group; it is defined by each maintainer device and informs each one whether the heaters related to the offices have a problem. Figure 8 describes the *BrandSupport* handler definition and also its association with the maintainer group of users. To define a handler in our mechanism it is necessary to extend the *Handler* abstract class. This class requires the implementation of *verifyContextCondition* and *execute* abstract methods defined in the API of our mechanism. The first method performs verification if the exceptional context is really appropriate for this handler and the second one executes the handler functionality. For instance, in our case study, each maintainer employee is responsible for a specific university region and also for a specific type of heating. For this reason, there is a handler for each employee with appropriate conditions. Therefore, when an exception is caught, the mechanism executes the *verifyContextCondition* for each handler defined in that scope. Whether this method returns *true*, the mechanism invokes *execute*, but if not, the mechanism follows to the next defined handler. The purpose of this approach is to promote extra flexibility that supports the definition of context-aware handlers. After the handler definition, Figure 8 depicts *UniversityHeaterFail* and the scope group definition. The exceptional context *UniversityHeaterFail* catch all *UnableToHeat* exception occurrences that come from *University* region and has one of the two brands (A or B). To deal with this exception, each maintainer device gets an instance of the *HeatMaintainer* scope group and adds itself to this scope. Thus, whenever *UnableToHeat* was propagated to the *HeatMaintainer*, each device can carry out the exceptional context through its context-aware handlers.

```

public class BrandSupport extends Handler {

public boolean verifyContextCondition(){
SimpleContext simple =getException().getContext().
    find("Computing Dep",
        "HeaterType = 'Ceramic'");
    if (simple != null) return true;
    else return false;
}
public boolean execute(){
    makeAppointment();
}
}

. . .
UniversityHeaterFail branduni =
    new UniversityHeaterFail(CompositeContext.OR);
branduni.addContext("University",
    "UnableToHeat.Brand='HeaterCompanyA'");
branduni.addContext("University",
    "UnableToHeat.Brand='HeaterCompanyB'");
BrandSupport suppCeramic = BrandSupport(branduni);
DeviceGroupScope groupScope = DeviceGroupScope.
    getInstance("HeatMaintainer");
groupScope.addDeviceList(this.getMyDefice());
groupScope.attachHandler(suppCeramic);
. . .

```

Figure 8. Group scope definition.

The second approach to handle the *UnableToHeat* is responsible to inform the fire brigade about a more dangerous situation that is potentially going on (Figure 9). This is done by mobile agents that have subscriptions in all maintainer groups. The agent defines *DangerousHeaterFail* exception that deals with the *University*

region, temperature and thermostat information which can ignite combustion. The mechanism needs to ensure that the temperature is really coming from the correct region when the environment contains a huge number of sensors that supply temperature information. For this reason, the user can define a combination of the constraints that compare the exception and temperature source region. This comparison does not use the LIS mechanism to support hierarchy regions, once we want the exact regions, not “super” regions, as for instance, University is equal to Computing Dep.

```

DangerousHeaterFail dangerfail = new
    DangerousHeaterFail(CompositeContext.AND);
dangerfail.addContext("University",
    "UnableToHeat.Thermostat='noanswer'");
dangerfail.addContext("University",
    "UnableToHeat.Region= Temperatura.Region");
dangerfail.addContext("University",
    "Temperatura.value > 30");
FireBrigade avoidfire = FireBrigade(dangerfail);
DeviceGroupScope groupScope = DeviceGroupScope.
    getInstance("HeatMaintainer");
groupScope.addDeviceList(
    getMyVirtualDevice());
groupScope.attachHandler(avoidfire);

```

Figure 9. Group scope for mobile agent.

To be aware of what is happening in the office, the user device need to define the exception shown in Figure 10. This exception represents all exception occurrences that come from its own current region. It is and associated with a handler that informs the user about the current problem.

```

BeAwareException awarex = new BeAwareException ();
Unaheat.addContext(device.getRegion());
NotifyUsers ntusers = new NotifyUsers(awarex);
RegionScope regionScope =
    RegionScope.getInstance();
regionScope.attachHandler(ntusers);

```

Figure 10. Region scope definition.

As we can see in Figure 7, if none of the devices that are part of the group scope do not handle the *UnableToHeat* exception, it will be propagated to the server scope. To deal with this exception, Figure 11 illustrates the exception and server scope definition. The *MakExternal* handler creates an external request to fix the problem that no internal maintainer is able to satisfy.

```

UnableToHeat uncatch = new UnableToHeat();
MakExternal makexternal =
    new MakExternal(uncatch);
ServerScope serverScope = ServerScope.
    getInstance("MainServer");
serverScope.attachHandler(makexternal);

```

Figure 11. Server scope definition.

Finally, Figure 12 depicts three dimensions in which our exception handling mechanism is applied: the application elements, the internal mechanism components, and the MoCa components. For example, the application programmer needs to defines application-specific exceptional contexts, add contextual information, define context-aware handlers, and attach them to the scopes. In MoCa exceptional contexts are defined in the CIS as a set of subscriptions and listeners supplied by the MoCa API.

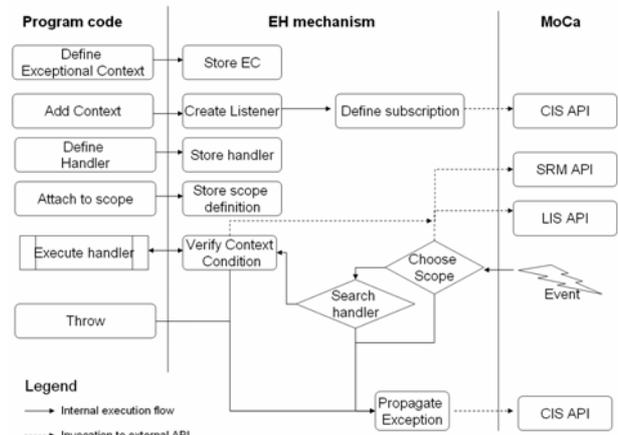


Figure 12. Different views of the exception mechanism

5. Discussions and Related Work

Although our current implementation (Section 4) supports a heterogeneous set of handling scopes, their granularity may not be always appropriate. To this end we are planning to adopt role-like abstractions, as supported by the CAMA tuplespace-based middleware [6], in order to allow handlers to be attached the specific agent actions or plans. Furthermore we plan to extend our mechanism to support code mobility in addition to physical mobility. In our previous work [18, 19], we have combined reflective and tuplespace middleware features (Section 2.1) in order to smoothly support code migration. We are also working on developing proactive exception handling (Section 3.5).

Developing advanced exception handling mechanisms suitable for multi agent systems is an area that needs serious efforts from the research community even though there have been a number of interesting results. A scheme in [17] supports exception handling in systems consisting of agents that cooperate by sending asynchronous messages. This scheme allows handlers to be associated with services, agents and roles, and supports concurrent exception resolution. Paper [22] identifies several typical failure cases in building context-based collaborative applications and proposes an exception handling mechanism for dealing with them.

An approach in [24] is based on defining a specialized service fully responsible for coordinating all exception handling activities in multi agent systems. Although this approach does not scale well, it supports separation of the normal system behavior from the abnormal one as the service carries all fault tolerance activities: it detects errors, finds the most appropriate recovery actions using a set of heuristics and executes them. As opposed to the last three schemes above which do not explicitly introduces the concept of the exception handling context (scope), the CAMA framework [6] (introduced in Section 2.1) supports the concept of (nested) scopes, which confine the errors and to which exception handlers are attached. However, CAMA and the other mechanisms mentioned above do not support a fully context-aware exception handling, as supported by our approach (Section 4). In particular, they do not implement context-aware selection of handlers, proactive exception handling, and the definition of exceptional contexts.

6. Final Remarks

Error handling in mobile agent-based applications needs to be context sensitive. This paper discussed our experience in incorporating exception handling in several prototype MoCA applications. This allowed us to elicit a set of requirements and define a novel context-aware exception handling model, which consists of: (i) explicit support for specifying “exceptional contexts”; (ii) context-sensitive search for exception handlers; (iii) multi-level handling scopes that meet new abstractions (such as groups), and abstractions in the underlying context-aware middleware, such as devices, regions, and proxy servers, (iv) context-aware error propagation, (v) contextual exception handlers, and (vi) proactive exception handling. We have also presented an implementation of this mechanism in the MoCA architecture, and illustrated its use in an AmI agent-based application.

7. REFERENCES

- [1] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *Proc. of the 2000 Conference on Human Factors in Computing Systems*, The Hague, The Netherlands, April 2000.
- [2] Abowd, G. et al. 1999. Towards a Better Understanding of Context and Context-Awareness. In *Proc. of the 1st Intl. Symp. on Handheld and Ubiquitous Computing* (Karlsruhe, September 1999, LNCS 1707. Springer, 304-307.
- [3] Dey, A. 2001. Understanding and Using Context. *Personal Ubiquitous Comput.* 5, 1 (Jan. 2001), 4-7.
- [4] Schilit, B. N, Adams, R. and Want R.. Context-aware computing applications. In *Proc. Workshop on Mobile Computing Systems and Applications*. IEEE, December 1994.
- [5] Davidyuk, O. et al. Context-aware middleware for mobile multimedia applications. In *Proc. of the 3rd international Conference on Mobile and Ubiquitous Multimedia* (College Park, Maryland, October 27 - 29, 2004). vol. 83. ACM Press, New York, NY, 213-220.
- [6] Iliasov, A. and Romanovsky, A. CAMA: Structured Communication Space and Exception Propagation Mechanism for Mobile Agents. *ECOOP-EHWS 2005*, 19 July 2005, Glasgow.
- [7] Vinod Muthusamy et al. Publisher Mobility in Distributed Publish/Subscribe Systems, Fourth International Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05), 2005.
- [8] Cugola, G. and Cote, J. E. M. "On Introducing Location Awareness in Publish-Subscribe Middleware," Fourth International Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05), 2005.
- [9] Gelernter, D. 1985. Generative communication in Linda. *ACM Trans. Program. Lang. Syst.* 7, 1 (Jan. 1985), 80-112.
- [10] Blair, G. et al.: An architecture for next generation middleware. In: *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*. London: Springer-Verlag, (1998).
- [11] Tripathi, A.: Challenges designing next-generation middleware systems. *Commun. ACM*, ACM Press, v. 45, n. 6, (2002), p. 39-42.
- [12] Smith, B. C.: *Procedural Reflection in Programming Languages*. These (Phd) —Massachusetts Institute of Technology, (1982).
- [13] Roman, M., F. Kon, and R.H. Campbell.: *Reflective Middleware: From Your Desk to Your Hand*. *IEEE Distributed Systems Online Journal*, 2(5), (2001).
- [14] V. Sacramento et al, "MoCA: A Middleware for Developing Collaborative Applications for Mobile Users," *IEEE Distributed Systems Online*, vol. 5, no. 10, 2004.
- [15] S. Sharples, V. Callaghan, G. Clarke, "A Multi-Agent Architecture for Intelligent Building Sensing and Control" *International Sensor Review Journal*, May 1999.
- [16] N. Shadbolt, "Ambient intelligence", *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 4, pp. 2-3, Jul. – Aug. 2003.
- [17] F. Souchon et al. Improving exception handling in multi-agent systems. In C. Lucena et al (Eds), *Software Engineering for Multi-Agent Systems II*, number 2940. Feb. 2004.
- [18] O. Silva, A. Garcia, C. Lucena. The Reflective Blackboard Pattern: Architecting Large-Scale Multi-Agent Systems. In: "Software Engineering for Large-Scale Multi-Agent Systems". Springer, LNCS 2603, April 2003, pp. 76-97.
- [19] O. Silva, A. Garcia, C. Lucena T-Rex: A Reflective Tuple Space Environment for Dependable Mobile Agent Systems. *Proc. IEEE Workshop on Wireless Communication and Mobile Computation (WCSF'01)*, Recife, Brazil, Aug 2001.
- [20] A.F. Garcia, C. M. F. Rubira, A. Romanovsky, J. Xu. A Comparative Study of Exception Handling Mechanisms for Building Dependable Object Oriented Software: *Journal of Systems and Software*. 59(2001), 197-222.
- [21] Goodenough, J. B. 1975. Exception handling: issues and a proposed notation. *Commun. ACM* 18, 12 (Dec. 1975), 683-696.
- [22] A. Tripathi, D. Kulkarni, T. Ahmed. Exception Handling Issues in Context Aware Collaboration Systems for Pervasive Computing. In Romanovsky, A., Dony, C., Knudsen, J. L., Tripathi, A. (Eds.) *Developing Systems that Handle Exceptions*. *Proc. ECOOP 2005 Workshop on Exception Handling in Object Oriented Systems*. TR 05-050. LIRMM. Montpellier-II University. 2005. July. France.
- [23] Capra, L. Emmerich, W. and Mascolo, C. CARISMA: Context-Aware Reflective Middleware System for Mobile Applications. *IEEE Transactions on Software Engineering* 29(10): pp. 929-944, Oct 2003.
- [24] M. Klein. C. Dellarcas. Exception Handling in Agent Systems. *Proc. of the 3rd Int. Conference on Autonomous Agents*, Seattle, WA, May 1-5, 1999. Pp. 62-6
- [25] T. Anderson and P. A. Lee. *Fault Tolerance: Principles and Practice*. Prentice-Hall, 2nd edition, 1990.

Rephrasing Rules for Off-The-Shelf SQL Database Servers

Illir Gashi, Peter Popov

Centre for Software Reliability, City University,
Northampton Square, London, EC1V 0HB
I.Gashi@city.ac.uk, ptp@csr.city.ac.uk

Abstract

We have reported previously [1] results of a study with a sample of bug reports from four off-the-shelf SQL servers. We checked whether these bugs caused failures in more than one server. We found that very few bugs caused failures in two servers and none caused failures in more than two. This would suggest a fault-tolerant server built with diverse off-the-shelf servers would be a prudent choice for improving failure detection. To study other aspects of fault tolerance, namely failure diagnosis and state recovery, we have studied the “data diversity” mechanism and we defined a number of SQL rephrasing rules. These rules transform a client sent statement to an additional logically equivalent statement, leading to more results being returned to an adjudicator. These rules therefore help to increase the probability of a correct response being returned to a client and maintain a correct state in the database.

1. Introduction

Fault tolerance is frequently the only viable approach of obtaining the required system dependability from systems built out of “off-the-shelf” (OTS) products [2]. There are various methods in which this fault tolerance can be achieved ranging from simple error detection and recovery add-ons (e.g. wrappers [3]) to diverse redundancy replication using diverse versions of the components.

These design solutions are well known. Questions remain, however, about the dependability gains and implementation difficulties for a specific system.

We have studied some of these issues in SQL database servers, a very complex category of off-the-shelf products. We have previously reported [1] results from a study with a sample of bug reports from four off-the-shelf SQL servers so as to assess the possible advantages of software fault tolerance - in the

form of modular redundancy with diversity - in complex off-the-shelf software. We found that very few bugs cause failures in two servers and none cause failures in more than two, which would indicate that significant dependability improvements can be expected from the deployment of a fault-tolerant server built out of diverse off-the-shelf servers in comparison with individual servers or the non-diverse replicated configurations.

Although we found that using multiple diverse SQL servers can dramatically improve error detection rates it does not make them 100%, e.g. our study [1] found four bugs causing identical non-self-evident failures in two servers. Thus there is room for improving failure detection further. Many of the cases, in which a failure was detected did not allow for immediate diagnosis of the failed server. Fault tolerance requires also diagnosing the faulty server and maintaining data consistency among the databases in addition to failure detection. To improve the situation, we studied the mechanism called “data diversity” by Ammann and Knight [4] (who studied it in a different context). The simplest example of the idea in [4] refers to computation of a continuous function of a continuous parameter. The values of the function computed for two close values of the parameter are also close to each other. Thus, failures in the form of dramatic jumps of the function on close values of the parameter can not only be detected but also corrected by computing a “pseudo correct” value. This is done by trying slightly different values of the parameter until a value of the function is calculated which is close to the one before the failure. This was found [4] to be an effective way of detecting as well as masking failures, i.e. delivering fault-tolerance. Data diversity, thus, can help with failure detection and state recovery, and thus complement fault-tolerance solutions which employ diverse modular redundancy, as well as helping achieve a certain degree of fault tolerance without employing diverse modular redundancy.

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

Data diversity is applicable to SQL servers because of the inherent redundancy that exists in the SQL language: statements can be “rephrased” into different, but logically equivalent [sequences of] statements. While working with the bug reports we found examples where a particular statement causes a failure in a server but a rephrased version of the same statement does not. Examples of such statements often appear in bug reports as “workarounds”.

In this paper we provide details of how SQL rephrasing can be employed systematically in a fault-tolerant server and provide examples of useful rephrasing rules. We also report on performance measurements using the TPC-C [5] benchmark client implementation to get some initial estimates of the delays introduced by rephrasing.

The paper is structured as follows: in section 2 we give details of the architecture of a fault-tolerant server employing rephrasing. In section 3 we give details of the data diversity study we have conducted for defining SQL rephrasing rules and illustrate how one of these rules has been used as a workaround for two known bugs of two SQL servers. In section 4 we give some empirical results of experiments we have conducted to measure the performance penalty due to rephrasing. In section 5 we discuss some general implications of our results and finally in section 6 some conclusions are presented with possibilities for further work.

2. Architecture of a Fault-Tolerant Server

2.1 General Scheme

Data replication is a well-understood subject [6], [18], [7]. The main problem replication protocols deal with is guaranteeing consistency between copies of a database without imposing a strict synchronisation regime between them. A study which compared various replication protocols in terms of their performance and the feasibility of their implementation can be found in [8]. Existing protocols implement efficient solutions for this problem, but depend on running copies of the *same* (non-diverse) server. These schemes would not tolerate non-self-evident¹ failures that cause incorrect writes to the database or

¹ In [1] we classified the failures according to their detectability by a client of the database servers into: *Self-Evident failures* - engine crash failures, cases in which the server signals an internal failure as an exception (error message) and performance failures; *Non-Self-Evident failures*: incorrect result failures, without server exceptions within an accepted time delay.

that return incorrect results from read statements. For the former, incorrect writes would be propagated to the other replicas and for the latter, incorrect results would be returned to the client. This deficiency can be overcome by building a fault-tolerant server node (“FT-node”) from two or more diverse SQL servers, wrapped together with a “middleware” layer to appear to each client as a single SQL server. An illustration of this architecture with two diverse Off-The-Shelf servers (“O-servers”) is shown in Fig. 1. A brief explanation of the figure follows. Several nodes (computers) are depicted which run client applications (Client node 1, Client node 2 and Client node 3) or server applications (Middleware node, RDBMS 1 node and RDBMS 2 node). The bottom three nodes together form the FT-server. Components may share a node: e.g. Replication Middleware, and the two SQL connectors for dialects 1 and 2 are deployed on the Middleware node. The SQL connectors additionally contain the SQL rephrasing rules. The diagram assumes that the Off-The-Shelf servers (O-servers) run on separate nodes, RDBMS 1 node and RDBMS 2 node. The circles represent the interfaces through which the components interact. Each SQL connector, implements the SQL Connector API interface used by the Replication Middleware component. This, in turn implements the Middleware API interface via which the client applications access the FT-server, either directly or via a driver for the FT-server in a specific run-time environment, e.g. JDBC driver or .NET Provider.

Further improvements to this architecture would be to also run diverse replicas of the middleware component. We have described elsewhere [9], [2] in more detail the FT-node architecture. Here we will only elaborate on the parts relevant to the discussion of rephrasing.

2.2 SQL Connectors

The O-servers are not fully compatible: they “speak different dialects” of SQL, despite being compliant at various levels with SQL standards. Therefore the FT-server includes a translator between these dialects, defined for a subset of SQL (e.g. “SQL-92 entry level”) plus some more advanced features important for enterprise applications (such as TRIGGERS and STORED PROCEDURES). The translators are depicted as “SQL Dialect Connector’s” in Fig 1.

A similar idea (implemented in [10], [11]) is to re-define the grammar of one database server to make it compatible with that of another while keeping the core database engine unchanged.

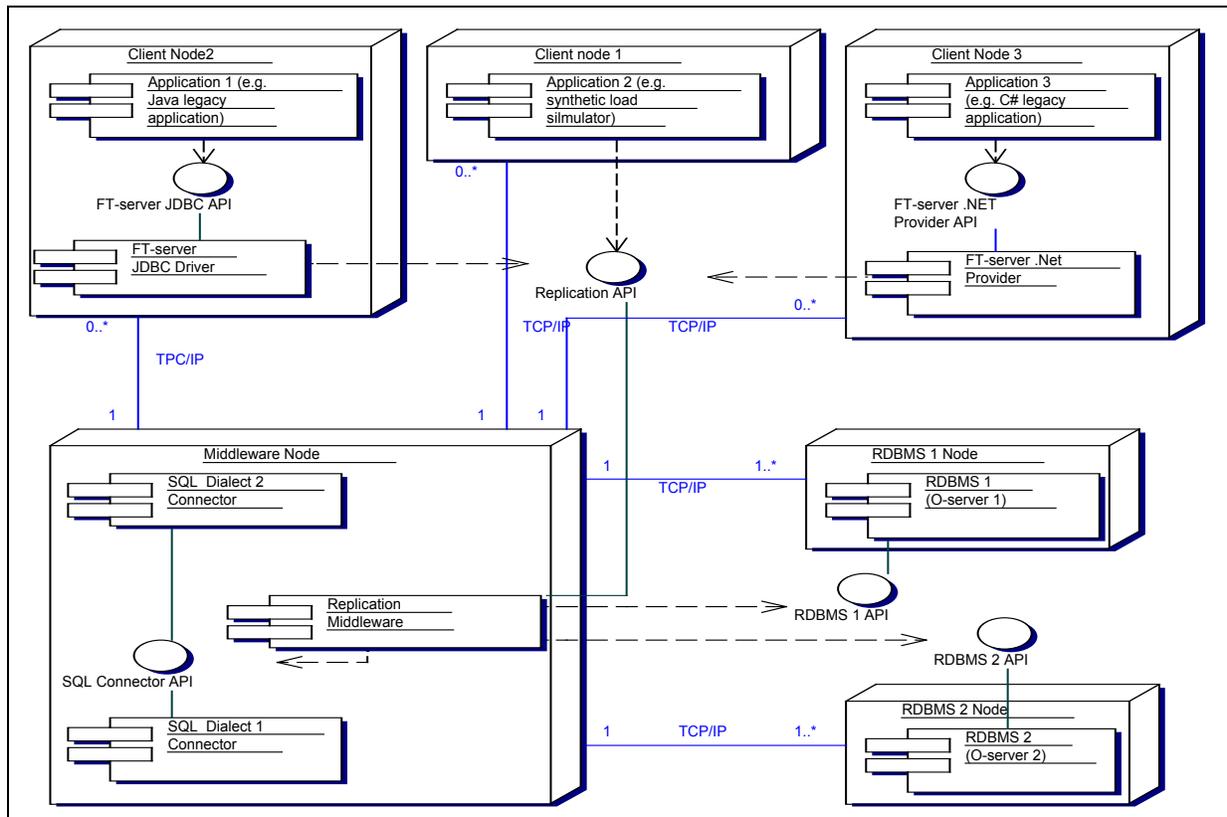


Fig. 1 - UML Deployment diagram of the FT-server.

2.3 Failure Detection, Masking, Recovery

The middleware of the FT-server includes extensive functionality for failure detection, masking and state recovery. Self-evident server failures are detected as in a non-diverse server, via server error messages (i.e. via the existing error detection mechanisms inside the servers), and time-outs for crash and performance failures. Diversity gives the additional capability of detecting non-self-evident failures by comparing the outputs² of the different O-servers. In a FT-node with 3 or more diverse O-servers, majority voting can be used to choose a result and thus mask the failure to the clients, and identify the failed O-server which may need a recovery action to correct its state. With a 2-diverse FT-node, if the two O-servers give different results, the middleware cannot decide which O-server is in error.

² An “output” may be the results from a SELECT statement or the number of rows affected for a write (INSERT, UPDATE and DELETE) statement. For INSERT and UPADTE statements a more refined way would be to read back the affected rows and use those for comparison.

This is where “data diversity” can help by providing additional results to break the tie (more in the next subsection). State recovery of the database can be obtained in the following ways:

- via standard backward error recovery, which will be effective if the failures are due to transient failures (caused by so called “Heisenbugs” [12]). To command backward error recovery, the middleware may use the standard database transaction mechanisms: aborting the failed transaction and replaying its statements may produce a correct execution. With “data diversity” a finer granularity level of recovery is possible using SAVEPOINTS and ROLLBACKS;
- additionally, diversity offers ways of recovering from non-transient failures (caused by so called “Bohrbugs” [12]), by essentially copying the database state of a correct server into the failed one (similarly to [13]). Since the formats of the database files differ between the servers, the middleware would need to query the correct server[s] for their database contents and command the failed server to write them into the corresponding records in its database, similar to what is proposed in [14]. This

would be expensive, perhaps to be completed off-line, but a designer can use multi-level recovery, in which the first step is to correct only those records that have been found erroneous on read statements.

2.4 Data Diversity Extensions

Even with just two diverse O-servers, many of the O-server failures may be masked by using “data diversity” (rephrasing an SQL statement into a different, but semantically equivalent one) to solicit “second opinions” from the O-servers and if possible outvote the incorrect response.

Data diversity could be implemented via an algorithm in the “Middleware Node” that rephrases statements according to predefined rules. We can define these rules for each type of SQL statement defined by the SQL grammar implemented by the server. These rules therefore may form part of the “SQL Dialect Connectors”. Upon receiving a statement from a client application the middleware can look up a rule from the list of available rules and rephrase the statement. The middleware must allow for new rules to be defined as and when necessary. If the middleware exhausts the list of rules that it can apply to a certain statement but no “correct result”³ can be established by applying the closed adjudication mechanism then an error message is returned to the client.

Data diversity can be used *with* or *without* design diversity. Architectural schemes using data diversity are similar to those using design diversity. For instance, Amman and Knight in [4] describe two schemes, which they call “retry block” and “n-copy programming”, which can also be used for SQL servers. The “retry block” is based on backward recovery. A statement is only rephrased if either the server “fail-stops” or its output fails an acceptance test. In “n-copy programming”, a copy of the statement as issued by the client is sent to one of the O-servers and rephrased variant(s) are sent to the others; their results are voted to mask failures.

Data diversity allows for a finer-granularity of state recovery, which is facilitated by the implementation of “SAVEPOINT” and “ROLLBACK” within transactions. The procedure (written in pseudocode), for a statement within a transaction, is given at the end of this subsection.

A performance optimization could be to perform adjudication at an intermediate step of the WHILE loop execution rather than at the end (e.g. for a “majority

voting” adjudication, if there are five rules for a particular statement then could check after the execution of the first three rephrased versions of the statement whether results returned by each of them are identical; if yes then majority result is already obtained and therefore no need for the last two rephrased versions of the statement to be executed).

The SAVEPOINT and ROLLBACK approach is the correct way of ensuring the “isolation” property of an ACID transaction.⁴ Otherwise, if we “ABORTed” the transaction and started a new one to perform the rephrased version of the statement, a concurrent transaction may have updated rows in the target table. This would lead to different results being returned by the O-server for the rephrased statement even though the behavior is not faulty.

```

WHILE more rephrasing rules available for the statement DO
  IF WRITE (i.e. DML (INSERT, UPDATE or DELETE) or DDL (e.g.
  CREATE VIEW etc.)) statement THEN
    SAVEPOINT;
    Execute WRITE statement[s] produced by the current re-
    phrasing rule;
    READ the rows amended by the WRITE statement;
    Store the results produced by the preceding READ state-
    ment;
    ROLLBACK TO last SAVEPOINT;
  ELSE IF READ (i.e. SELECT) statement THEN
    Execute READ statement[s] produced by the current re-
    phrasing rule;
    Store the results produced by the READ statement;
  END IF
END WHILE
Adjudicate from the stored results produced by each rephrased version
of the statement;
IF adjudication succeeds (e.g. “majority voting” produced a result) THEN
  Execute the statement which was adjudicated to be correct;
ELSE
  ABORT current Transaction
  Raise an exception;
END IF

```

3. SQL Rephrasing Rules

As explained in section 2, the support for data diversity can be implemented in the middleware in the form of rephrasing rules. The initial step is defining the rules that are to be implemented. The rules can be defined by studying in depth the SQL language itself to identify the parts of the language which are synonymous and therefore enable the definition of logically equivalent rephrasing rules. We took a different more

³ Depending on the setup used a correct result could be either the majority result or one that passes an acceptance test.

⁴ This is under the assumption that the ACID property of the transaction is failure-free.

direct approach to defining these rules: we studied the known bugs reported for 4 open-source servers, namely Interbase 6.0, Firebird 1.0⁵, PostgreSQL 7.0 and PostgreSQL 7.2 (abbreviated IB 6.0, PG 7.0, FB 1.0 and PG 7.2 respectively). However our intention was not to simply define workaround rules which are highly bug specific, but instead to define generic rephrasing rules, which can be used in a broader setting. As a result we found that some of the generic rules that we defined could be applied to multiple bugs in our study. We provide examples next.

3.1 Generic Rules

The “generic rules” are rephrasing rules, which can be applied to a range of ‘similar’ statements, be it DML (data manipulation language: SELECT, INSERT, UPDATE and DELETE) or DDL (data definition language e.g. CREATE TABLE etc.) statements. We have defined a total of 14 generic rephrasing rules. Full details of these rules are in [15]. We will provide details of Rule 8 and how it proved to be a useful *workaround* for two different bugs reported for two different servers.

Rule 8: *An SQL VIEW can be rephrased as an SQL STORED PROCEDURE or SQL TEMPORARY TABLE*

This rule proved to be a useful workaround for FB 1.0 Bug 488343 [16]. To observe the failure the bug report details the following setup:

```
CREATE TABLE CUSTOMERS (ID INT, NAME, VARCHAR(10));
CREATE TABLE INVOICES (ID INT, CUST_ID INT, CODE
    VARCHAR(10), QUANTITY INT);
INSERT INTO CUSTOMERS VALUES (1, 'ME');
INSERT INTO INVOICES VALUES (1, 1, 'INV.1', 5);
INSERT INTO INVOICES VALUES (2, 1, 'INV.2', 10);
INSERT INTO INVOICES VALUES (3, 1, 'INV.3', 15);
INSERT INTO INVOICES VALUES (4, 1, 'INV.4', 20);
```

The following VIEW is faulty (specifically, the use of the SQL DISTINCT keyword to filter the results of a SELECT statement is faulty in SQL VIEWS of the FB 1.0 server):

```
CREATE VIEW V_CUSTOMERS AS SELECT DISTINCT ID, NAME
    FROM CUSTOMERS;
```

The failure can be observed by issuing the following statement:

```
SELECT SUM(INV.QUANTITY) FROM INVOICES INV INNER JOIN
    V_CUSTOMERS CUST ON INV.CUST_ID = CUST.ID;
```

SUM
20

⁵ Firebird is the open-source descendant of Interbase 6.0. The later releases of Interbase are issued as closed-development by Borland.

The expected result is 50 not 20. If we use a STORED PROCEDURE instead of the VIEW then the correct results is returned⁶:

```
SET TERM !!;
CREATE PROCEDURE V_CUSTOMERS RETURNS (ID INT, NAME
    VARCHAR(10)) AS
BEGIN
    FOR SELECT DISTINCT ID, NAME FROM CUSTOMERS
        INTO :ID, :NAME DO
        BEGIN
            SUSPEND;
        END
    END
END!!
SET TERM; !!
```

Issuing the same SELECT statement as before we obtain the expected result (50):

```
SELECT SUM(INV.QUANTITY) FROM INVOICES INV INNER JOIN
    V_CUSTOMERS CUST ON INV.CUST_ID = CUST.ID;
```

SUM
50

The same rule was a useful workaround for another bug, this time the PG 7.0 bug 23 [17]. To observe the failure the bug report details the following setup:

```
CREATE TABLE L (PID INT NOT NULL, SEARCH BOOL, SERVICE
    BOOL);
INSERT INTO L VALUES (1,'T','F'); INSERT INTO L VALUES (1,'T','F');
INSERT INTO L VALUES (1,'T','F'); INSERT INTO L VALUES (1,'T','F');
INSERT INTO L VALUES (1,'T','F'); INSERT INTO L VALUES (1,'F','F');
INSERT INTO L VALUES (1,'F','F'); INSERT INTO L VALUES (2,'F','F');
INSERT INTO L VALUES (3,'F','F'); INSERT INTO L VALUES (3,'T','F');
```

The following VIEWS are then defined (notice the use of the GROUP BY clause):

```
CREATE VIEW CURRENT AS SELECT PID, COUNT(PID), SEARCH,
    SERVICE FROM L GROUP BY PID, SEARCH, SERVICE;
CREATE VIEW CURRENT2 AS SELECT PID, COUNT (PID),
    SEARCH, SERVICE FROM L GROUP BY PID, SEARCH, SERVICE;
```

By issuing the following SELECT statement incorrect results are obtained (this is due to the GROUP BY clause used in the VIEWS and the COUNT used on a column from a VIEW):

```
SELECT CURRENT.PID, CURRENT.COUNT AS SEARCHTRUE,
    CURRENT2.COUNT AS
    SEARCHFALSE FROM CURRENT,CURRENT2 WHERE
    CURRENT.PID =CURRENT2.PID AND CURRENT.SEARCH='T'
    AND CURRENT2.SEARCH='F' AND CURRENT.SERVICE='F' AND
    CURRENT2.SERVICE='F';
```

-- pid | searchtrue | searchfalse

```
-- 1 | 10 | 10
-- 3 | 1 | 1
```

The expected results are:

-- pid | searchtrue | searchfalse

```
-- 1 | 5 | 2
-- 3 | 1 | 1
```

⁶ The syntax used is specific for Firebird.

By using TEMPORARY TABLEs instead of VIEWs the correct result is obtained:

```
SELECT PID, COUNT(PID), SEARCH, SERVICE INTO TEMP
CURRENT FROM L GROUP BY PID, SEARCH, SERVICE;
SELECT PID, COUNT(PID), SEARCH, SERVICE INTO TEMP
CURRENT2 FROM L GROUP BY PID, SEARCH, SERVICE;
SELECT CURRENT.PID,CURRENT.COUNT AS SEARCHTRUE,
CURRENT2.COUNT AS SEARCHFALSE FROMCURRENT,
CURRENT2 WHERE CURRENT.PID=CURRENT2.PID AND
CURRENT.SEARCH='T' AND CURRENT2.SEARCH='F' AND
CURRENT.SERVICE='F' AND CURRENT2.SERVICE='F';
```

```
-- pid | searchtrue | searchfalse
-- 1 | 5 | 2
-- 3 | 1 | 1
```

We used TEMPORARY TABLEs in PG 7.0 and not STORED PROCEDUREs since PG 7.0 does not support functions (procedures) that return multiple rows.

Details of the other generic rephrasing rules and how they can be used as workarounds for other reported bugs are given here [15].

We looked at how many of the generic rules can be applied to the bugs reported for the open-source servers in our bugs study. The results are shown in Table 1. The leftmost three columns of the table show the results for the non-self-evident failures caused by read (i.e. SELECT) statements. Clearly, a number of these are also classified as a “user error”, i.e. the user issues an incorrect statement, which the server incorrectly executes without raising an exception. For example IB 6.0 incorrectly executes a statement such as SELECT X FROM A, B even though the column X is defined in both tables A and B, which can lead to ambiguous results. PG 7.0 / PG 7.2, correctly, raise an exception.

If we take away the “user error” bugs then we can see that in all the server pairs the generic rules can be used as workarounds for at least 80% of the non-self-evident failures caused by read statements.

The right-most 4 columns of the table are for the bugs that cause state-changing failures, which have been further subdivided into bugs in DDL and write statements. We can see that generic rules can be used as workarounds for at least 60% of failures caused by the state-changing statements.

3.2 Specific Rules

The generic rephrasing rules that we have defined do not provide workarounds for all the failures caused by the bugs collected in our study. For these failures specific workaround rules need to be defined. For example recursive BEFORE UPDATE TRIGGERS can return error messages in FB 1.0/IB 6.0 which means the table for which the trigger is defined becomes unusable (FB 1.0 bug 625899 [16]). A generic rule could not be defined for this bug. A specific workaround (and a generic recovery procedure) upon encountering this error message would be to:

- disable the trigger in FB 1.0 / IB 6.0
- read the log of the other server to check the sequence of the write statements that have been issued as a result of the trigger
- send this sequence of statements explicitly to the FB 1.0 / IB 6.0 server

The workaround above would work in a diverse server-type configuration if the other server[s] works correctly (the other server[s] in our study do not contain this bug) while without design diversity a fault, clearly, cannot be dealt with this way.

We have found that a large number of bugs, if server diversity is not employed, would require very specific rules to be defined to workaround the failures that they cause. In many cases these rules require substantial new implementation in the form of “wrapping” of the results returned to the client (or for write statements before they are stored in the database) or re-implementing parts of the functionality of the database that are found to be faulty and no workaround exists in SQL. Although possible such an approach is clearly limited because the newly developed code can itself be faulty which may diminish the gains in reliability that can be obtained from its use. This reiterates that design diversity is desirable.

4. Performance Implications of Rephrasing

To measure the performance implications of rephrasing, we conducted a number of experiments based

Table 1. A summary of applying the generic rephrasing rules for non-self evident and state-changing bugs of IB 6.0 and PG 7.0 and the later releases FB 1.0 and PG 7.2

Server pair	Non-self evident non-state-changing failures (SELECT statements)			State-changing failures			
				DDL statement failures		Write statement failures	
	Total	Total covered by generic rules	Total user errors *	Total	Total covered by generic rules	Total	Total covered by generic rules
IB 6.0 + PG 7.0	21	12	6	21	13	9	7
IB 6.0 + PG 7.2	26	18	6	19	13	7	5
FB 1.0 + PG 7.0	16	11	2	19	13	8	6
FB 1.0 + PG 7.2	19	15	2	17	13	6	4

on the industry standard benchmark for databases - TPC-C [5]⁷. The factors which degrade performance when rephrasing is employed are:

1. delays enforced by the middleware for comparison of results
2. delays from using the following mechanisms within transactions:
 - Transaction SAVEPOINTS
 - Transaction ROLLBACKS
 - Execution of SELECT statements after WRITE statements (INSERT, UPDATE, DELETE)
 - Rephrasing

The additional delay introduced by the use of rephrasing is delay 2. We have performed an experimental study to estimate delay 2. Delay 1 would exist also in a diverse setup with or without rephrasing. Studies that have reported measures of other delays which are not specific to rephrasing (such as enforcing 1-copy serialisability) can be found in [18], [6]⁸. There are other factors that can influence the degradation of performance that we have not measured in our experimental setup (e.g. rephrasing delays when more than one rephrasing rule is used etc.). The experiments that we have conducted aim to provide an initial estimate of the delays due to rephrasing. A more thorough performance evaluation should also take into account concurrent execution of transactions. As was also noted by one of the anonymous reviewers, for some concurrency control mechanisms, the increase in transaction execution times due to the use of rephrasing, the probability of conflicts due to concurrency may also increase which may further degrade performance.

The experimental setup consisted of three computers. All three computers ran on Microsoft's Windows 2000 operating system, they had 384 MB RAM, and Intel Pentium 4 1.5GHz processors. One machine hosted the client implementation of the TPC-C benchmark. The other two machines hosted the servers (PostgreSQL 8.0 and Firebird 1.5). We used later releases of the servers than the ones used in our bugs study since these earlier releases do not support SAVEPOINTS and ROLLBACKS within transactions. We have not used any commercial servers in our experiments since the license agreements are very restrictive with regard to publishing performance data.

We ran experiments on both diverse and non-diverse setups. In the diverse experiments we always wait for the slowest server response before we can start

⁷ The TPC-C experiments were carried out with 1 emulated client and 1 warehouse with client think times set to 0.

⁸ These studies also provide some optimisation procedures for 1-copy serialisability.

the next transaction. Therefore the diverse setups here are always slower (other configurations are possible and we have discussed some of these in [9]).

Figure 2 illustrates the sequence of executions within a transaction for the different non-diverse setups. The grey boxes represent the fault tolerance mechanism used whereas the dotted lines represent the added delay from the use of the respective mechanism. Setup a) is the baseline, against which we will measure the added delays. Setups b), c), and d) measure the delays of using the fault tolerance mechanisms when no failures are observed (i.e. the cost of being cautious)⁹. Setups e) and f), measure the cost of re-execution of a statement¹⁰. These experiments measure delays for a number of situations:

- re-execution of an unchanged statement as a possible protection against transient failures (caused by the so called "Heisenbugs" [12])
- re-execution of a logically equivalent rephrased statement in case the first one has failed self-evidently (i.e. a crash or other exceptional failures)
- re-execution of a logically equivalent rephrased statement to get additional results for comparison on the middleware to increase the likelihood of failure detection for non-self-evident failures

In our experiments we did not use rephrased statements. Instead, the same statement was executed twice. This is a simplification due to the absence of a proper implementation of rephrasing. In the absence of any other data, we wanted to get an initial estimate of the delays that the various fault tolerance mechanisms will produce with the database servers.

The diverse setups have a similar structure. The only difference is that in diverse setups we only use 1 SAVEPOINT (at the beginning of the transaction) rather than before each write statement and therefore we may also have only one ROLLBACK (at the end of transaction). For setups e) and f), this means that we first execute every statement once then we ROLLBACK to the beginning and execute all the statement again. So the difference between the diverse and non-diverse setups is a different level of granularity of using SAVEPOINTS/ROLLBACKS.

⁹ b) detection of erroneous writes; c) SAVEPOINT are used before write statements for finer grained recovery; d) both SAVEPOINTS are used and the modified rows are read back (combination of b) and c));

¹⁰ e) optimistic (on writes) rephrasing: each statement is executed twice; to ensure that the state of the database remains unchanged during the second execution of the write statement we use SAVEPOINTS and ROLLBACKS; f) pessimistic rephrasing: same as e) but the written rows are also read to protect against erroneous writes.

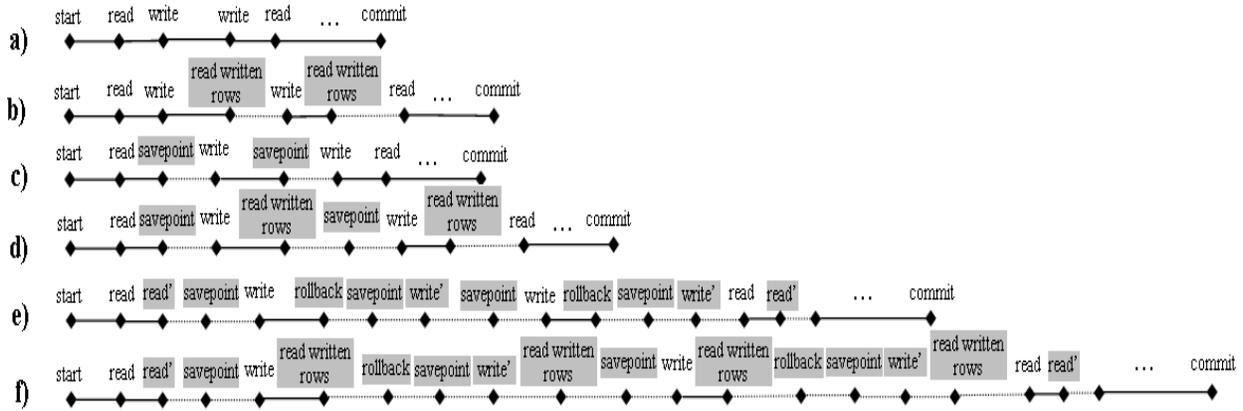


Fig. 2. A transaction execution sequence in the experimental setups. The shaded boxes represent the fault tolerant mechanism used and the dotted lines represent the additional delays from their use. The second executions of the statements are proxies for rephrased versions of statements.

The full results of these experiments are given in Table 2. The first column explains the setup under which the experiment was run. The following 4 columns spell out which fault tolerance mechanisms were used (if the cell is blank then the respective mechanism was not used). The following 3 columns show the average execution time of a transaction, and the last 3 columns show the added delay (in percentages) proportional to the baseline of each setup. The first six rows contain the results for each of the setups we explained earlier (and illustrated in Figure 2).

The last two rows are structurally the same as setups (e) and (f) respectively. However in these experiments we have tried to simulate the effect of a simple learning rule: if after 1000 executions a statement has been found to be correct then we stop rephrasing (in our simulation it means we stop executing the statement twice for both setups and additionally stop executing

the SELECT statement that read the modifications of the write statements for setup (h)).

The delays seem to be higher proportionally in PostgreSQL than in Firebird. This is because the execution time of COMMITs is smaller in Firebird for the experiments with larger number of SELECT statements. The number of write statements to be COMMITed always remains the same in all experiments (even in the ones with 2 executions of statements, since the first execution of a write statement is always ROLLBACKed). Comparing the setups a) with e) we can see that even though in setup e) every statement is being executed twice the average execution times of the transactions are not simply twice the execution time of transactions in setup a). This is explained by the fact that the number of transactions remains the same (i.e. we still have the same number of COMMITs) and also the data may be stored already in the RAM which reduces the execu-

Table 2 Performance effects of the various fault-tolerance schemes. Each experiment is run with loads of 10,000 transactions

Setup	Average Transaction Execution time (milliseconds)				Delays proportional to the baseline (%)		
	PG 8.0	FB 1.5	Diverse PG 8.0 & FB 1.5	PG 8.0	FB 1.5	Diverse PG 8.0 & FB 1.5	
Baseline (a)							
Detection of erroneous writes (b)				28.3	16.3	26.5	
Finer granularity of recovery (c)				5.2	0.4	1.0	
Combination of b and c (d)				33.9	18.6	26.0	
Optimistic (or pessimistic) Rephrasing (e)				54.0	46.0	42.2	
Pessimistic Rephrasing (f)				118.	96.2	105.5	
Learning Optimization (g)				12.6	6.0	17.2	
Learning Optimization (h)				22.5	11.4	52.6	

tion time of the second statement. The same holds when comparing results of setups b) with f).

Since the numbers in Table 2 represent point estimates (i.e. they are single runs of an experiment per setup) we have repeated the experiments for setup a) and f) to measure the non-deterministic variation that may exist between the different runs. We observed a very small difference (less than 1% for 5 out of six of the experiments and less than 3% for all). Hence we can trust with a higher degree of confidence that the observations documented in table 2 represent closely the ‘true’ differences between different setups.

5. Discussion

We presented in section 2 the architecture we propose for a fault-tolerant server employing rephrasing. The middleware used would make use of a rephrasing algorithm. Any fault-tolerant solution, which makes use of server diversity would need to have “connectors” developed as part of the middleware to translate a client sent statement to the dialect of the respective server. This is because each server ‘speaks’ its own dialect of SQL. The rephrasing algorithms can also be part of these connectors. A related point is that database servers offer features that are extensions to the SQL standard, and these features may differ between the servers. Therefore for applications which require a richer set of functionality data diversity would be attractive alone as it would for instance allow applications to use the full set of features. A complex statement, which can be directly executed with some servers but not others, may need to be rephrased as a logically equivalent sequence of simpler statements for the latter. For example, the TRUNCATE command is a PostgreSQL specific feature (and is buggy in version 7.0; see bug 20 [17] for details). In its stead the DELETE command can be used to workaround the problem. The DELETE command is also implemented in Firebird and all the other SQL compliant servers.

Since most of these rules are transformations of the SQL grammar, they are amenable to formal analysis. Thus, despite the additional implementation, high reliability can be achieved with a combination of formal analysis and testing of the new code.

The results presented in section 3 demonstrate that a small number of rephrasing rules can help with server diagnosis and state recovery. We observed that a limited set of generic rephrasing rules that we have defined (14 in total) can be used as workarounds for at least 80% of the non-self-evident failures caused by read statements and at least 60 % of failures caused by write or DDL statements in any of the open-source 2-diverse

setups in our study. We have also observed that using data diversity without design diversity would lead to a large number of *specific rephrasing rules* to work-around certain failures. Implementing such rules might require a substantial amount of new implementation, which itself may be faulty, thus, reducing the possible reliability gains that can be obtained from their use.

Rephrasing has been proposed as a possibility to detect failures that would otherwise be un-detectable in some replication settings. The possible benefits of this approach could be its relatively low cost in comparison with design diversity, and also that it can be used with or without design diversity allowing for various cost-dependability trade-offs. Possible setups include:

- In non-diverse redundant replication settings, if high dependability assurances are required, the only option available would be to rephrase all the statements sent to the server. This can lead to high performance penalties. To reduce the performance penalty some form of learning strategy can be applied, e.g. keep track of all the statements that have been rephrased. If the rephrased statement keeps giving the same results as the original statement then confidence is gained that the original statement is giving the correct result and the statement does not have to be rephrased in future occurrences (what we did in setups g) and h) of the TPC-C experiments). The other dimension is to stop sending the client-version of the statement to a server if it always gives an incorrect result. In this case the middleware can flag each occurrence of this statement and use the rephrased version of it without sending the original statement to the server [2]. This reduces the time taken to respond to the client.
- In a diverse server configuration a less rephrasing-intensive approach may be used where only the read statements (i.e. SELECTs) that return different results are rephrased (assuming that at least two servers are running in parallel so that a mismatch is detected). The rephrasing is also done for all the write statements (to ensure that the state of the database is not corrupted). Since a smaller set of statements needs to be rephrased the performance is enhanced. The non-self-evident identical failures, however, (we observed 4 of these in the study with known bugs of SQL servers [1]) will not be detected. To further enhance the performance the same learning strategies can be used as in the previous setup.

6. Conclusions

We have reported previously [1] on the dependability gains that can potentially be achieved from deploy-

ing a fault-tolerant SQL server, which makes use of diverse off-the-shelf SQL servers. From studying bugs reported for four off-the-shelf servers we reported that failure detection rates in 1-out-of-2 configurations was at least 94% and this increased to 100% in configurations which employed more than two servers. However fault tolerance is more than just failure detection. In this paper we reported on the mechanism of data diversity and its application with SQL servers in aiding with failure diagnosis and state recovery. We have defined 14 generic ‘workaround rules’ to be implemented in a ‘rephrasing’ algorithm which when applied to a certain SQL statement will generate logically equivalent statements. We have also argued that since these rules are transformations of the SQL language syntax, they are amenable to formal analysis and dependability gains from employing rephrasing are achievable despite the development of a bespoke new code.

We also outlined a possible architecture of a fault tolerant server employing diverse SQL servers and detailed how the middleware used in it can be extended to also handle rephrasing of SQL statements.

We also presented some performance measurements from experiments we have run with an implementation of the TPC-C benchmark [5], which gave initial estimates of the likely delays due to employing rephrasing.

Further work that is desirable includes:

- demonstrating the feasibility of automatic translation of SQL statements from, say ANSI/ISO SQL syntax to the SQL dialect implemented by the deployed SQL servers. We have completed some preliminary work on implementing translators between MSSQL and Oracle dialects for SELECTs, and between Oracle and PostgreSQL dialects for SELECT, INSERT and DELETE statements;
- developing the necessary components so that users can try out diversity in their own installations, since the main obstacle now is the lack of popular off-the-shelf “middleware” packages for data replication with diverse SQL servers. This would also include implementing a mechanism of maintaining (adding/removing) rephrasing rules as add-on components in the middleware.

Acknowledgment

This work has been supported in part by the Interdisciplinary Research Collaboration in Dependability (DIRC) project funded by the U.K. Engineering and Physical Sciences Research Council (EPSRC). Authors would like to acknowledge the anonymous reviewers for the thoughtful comments and useful suggestions.

Bibliography

1. Gashi, I., Popov, P., Strigini, L. *Fault diversity among off-the-shelf SQL database servers* in DSN'04, 2004, Florence, Italy, IEEE Computer Society Press p. 389-398.
2. Popov, P., et al. *Software Fault-Tolerance with Off-the-Shelf SQL Servers* in ICCBSS'04, 2004, Redondo Beach, CA USA, Springer p. 117-126.
3. Popov, P., et al. *Protective Wrapping of OTS Components in 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction*, 2001, Toronto
4. Ammann, P.E. and J.C. Knight, *Data Diversity: An Approach to Software Fault Tolerance*, IEEE Transactions on Computers, 1988, C-37(4), p. 418-425.
5. TPC, *TPC Benchmark C, Standard Specification, Version 5.0*. 2002 <http://www.tpc.org/tpcc/>.
6. Patiño-Martinez, M., Jiménez-Peris, R., Kemme, B., and Alonso, G., *MIDDLE-R: Consistent database replication at the middleware level*, ACM Transactions on Computing Systems, 2005, 23(4), p. 375-423.
7. Bernstein, P.A., V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. 1987, Reading, Mass.: Addison-Wesley. 370.
8. Jimenez-Peris, R., M. Patino-Martinez, G. Alonso, and B. Kemme, *Are Quorums an Alternative for Data Replication?*, ACM Transactions on Database Systems, 2003, 28(3), p. 257-294.
9. Gashi, I., Popov, P., Stankovic, V., Strigini, L., *On Designing Dependable Services with Diverse Off-The-Shelf SQL Servers*, in *Architecting Dependable Systems II*, R. de Lemos, C. Gacek, A. Romanovsky (Eds). 2004, Springer-Verlag. p. 191-214.
10. EnterpriseDB, *EnterpriseDB*. 2006 <http://www.enterprisedb.com/>.
11. Janus-Software, *Fyrracle*. 2006 http://www.janus-software.com/fb_fyrracle.html.
12. Gray, J. *Why do computers stop and what can be done about it?* in *6th International Conference on Reliability and Distributed Databases*, 1987
13. Tso, K.S. and A. Avizienis. *Community Error Recovery in N-Version Software: A Design Study with Experimentation* in FTCS-17, Pittsburgh, Pennsylvania, July 6-8, 1987 p. 127-133.
14. Sutter, H., *SQL/Replication Scope and Requirements document*, in *ISO/IEC JTC 1/SC 32 Data Management and Interchange WG3 Database Languages*. 2000. p. 7
15. Gashi, I., *Rephrasing Rules for SQL servers*. 2006 <http://www.csr.city.ac.uk/people/ilir.gashi/Bugs/>.
16. Gashi, I., *Tables containing known bug scripts of Firebird 1.0 and PostgreSQL 7.2*. 2005 <http://www.csr.city.ac.uk/people/ilir.gashi/Bugs/>.
17. Gashi, I., *Tables containing known bug scripts of Interbase, PostgreSQL, Oracle and MSSQL*. 2003 <http://www.csr.city.ac.uk/people/ilir.gashi/DSN/>.
18. Lin, Y., Kemme, B. et al., *Middleware based Data Replication providing Snapshot Isolation* in *ACM SIGMOD Int. Conf. on Management of Data*, 2005, Baltimore, Maryland, USA, ACM Press p. 419-430.

Fault Tolerance via Diversity for Off-The-Shelf Products: a Study with SQL Database Servers

Ilir GASHI, Peter POPOV, and Lorenzo STRIGINI, *Member, IEEE*

Abstract— If an off-the-shelf software product exhibits poor dependability due to design faults, software fault tolerance is often the only way available to users and system integrators to alleviate the problem. Thanks to low acquisition costs, even using multiple versions of software in a parallel architecture, a scheme formerly reserved for few and highly critical applications, may become viable for many applications. We have studied the potential dependability gains from these solutions for off-the-shelf database servers. We based the study on the bug reports available for four off-the-shelf SQL servers, plus later releases of two of them. We found that many of these faults cause systematic, non-crash failures, a category ignored by most studies and standard implementations of fault tolerance for databases. Our observations suggest that diverse redundancy would be effective for tolerating design faults in this category of products. Only in very few cases would demands that triggered a bug in one server cause failures in another one, and there were no coincident failures in more than two of the servers. Use of different releases of the same product would also tolerate a significant fraction of the faults. We report our results and discuss their implications, the architectural options available for exploiting them and the difficulties that they may present.

Index Terms— C.4.b Fault tolerance, C.4.f Reliability, availability, and serviceability, H.2.4.i Relational databases, D.2.17.e Error processing, design diversity, COTS software, fault records, non-crash failures, database availability.



1 INTRODUCTION

THE use of “off-the-shelf” (OTS) – rather than custom-built – products is attractive in terms of acquisition costs and time to deployment but brings concerns about dependability and “total cost of ownership”. For safety- or business-critical applications, in particular, purpose-built products would normally come with extensive documentation of good development practice and extensive verification and validation; when switching to mass-distributed OTS systems, users – system designers or end users – often find not only a lack of this documentation, but anecdotal evidence of serious failures and/or bugs that undermines trust in the product. Despite the large-scale adoption of some products, there is usually no formal statistical documentation of achieved dependability levels, from which a user could attempt to extrapolate the levels to be achieved in his/her own usage environment.

For all these reasons, when systems are built out of OTS products, software fault tolerance is often the only viable way of obtaining the required system dependability [1], [2], [3]. We use the phrase “software fault tolerance” to mean “fault tolerance against software faults”. These preliminary considerations apply not only to OTS software, but also to hardware, like microprocessors, or complete hardware-plus-software systems; but our focus in this paper is a category of software products. Fault tolerance may take multiple forms [4], from simple error detection and recovery add-ons (e.g. wrappers) [5] to full-fledged “diverse modular redundancy” (e.g. “N-version programming”: replica-

tion with diverse versions of the components) [4]. Even this latter class of solutions becomes affordable with many OTS products and has the advantage of a fairly simple architecture. The cost of procuring two or even more OTS products (some of which may be free) would still be far less than that of developing one’s own.

All these design solutions are well known from the literature. The questions, for the developers of a system using OTS components, are about the dependability gains, implementation difficulties and extra cost that they would bring for that specific system. We report here some evidence about potential gains, and briefly discuss the architectural issues that would determine feasibility and costs, for a specific category of OTS products: SQL database servers, or “database management systems” (DBMSs)¹.

This category of products offers a realistic case study of the advantages and challenges of software fault tolerance in OTS products. DBMS products are complex, mature enough for widespread adoption, and yet with many faults in each release². Fault tolerance in DBMS products is a thoroughly studied subject, with standard recognized solutions, some of which are commercially available. But these solutions do not give full protection against software faults, because they assume fail-stop [8] or at least self-evident

¹ Ordinary terms may be ambiguous when discussing redundant and diverse architectures. We will apply these conventions: a *DBMS product* is a specific software package; a fault-tolerant database server includes one or more *channels* (each performing the database server function), each including an installation of a DBMS product (these may be the same product or different ones - different *versions*) and a *replica* of the database. Two replicas of the database will be physically different if they are in channels that use different DBMS products. They may also exhibit temporary differences due to the asynchronous operation of the channels. We follow the popular usage of the word “bug” as synonym for “software fault” or “defect”.

² And even features that imply an accepted possibility of an incorrect behavior, albeit rare. An example of the latter is the known “write skew” [6] problem with some optimistic concurrency control architectures [7]

This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible.

• I. Gashi, P. Popov and L. Strigini are with Centre for Software Reliability, City University, Northampton Square, London EC1V 0HB, UK (telephone: + 44 20 7040 {0273, 8963, 8245}, e-mail: {ec233, ptp, strigini}@csr.city.ac.uk).

failures³: errors are detected promptly enough that the database contents are not corrupted, or that a suitable correct checkpoint can be identified and used for rollback. There is no guarantee that software faults in the OTS DBMS products themselves will satisfy this assumption. As we document here, they do not, and we know of no published statistical evidence of the frequency of violations, which one could use as evidence that the assumption is satisfied with high enough probability for a specific application of one of these OTS products.

There are many OTS SQL DBMS products, obeying (at least nominally) common standards (SQL 92 and SQL 99) that make diverse redundancy feasible in principle. For instance, a parallel-redundant architecture using two replicas of a database, managed by two diverse DBMS products, would allow error detection via comparison of results from the two DBMS products. A fault-tolerant server capable of tolerating server software faults can be built from installations of two or more diverse DBMS products, connected by middleware that makes them appear to clients as a single database server. There are clearly problems as well: in particular, existing DBMS products have certain concurrency control and fault tolerance features that rely on lack of diversity between replicated executions for their proper and efficient operation. However, it is worth exploring the costs and benefits of solutions that accept the drawbacks of diversity in return for improved dependability. For many users, there is no practical alternative to OTS DBMS products, and performance losses may well be acceptable in return for improved assurance. In addition to tolerating faults in general, users may look at software fault tolerance as a way of guaranteeing good service during upgrades of the DBMS products, when new bugs might appear that are serious under the usage profile of their specific installation, and/or of delaying “patches” and upgrades, thus reducing the total cost of ownership of DBMS products.

As a preliminary assessment of the potential effectiveness of software fault tolerance with DBMS products, we have studied publicly available fault reports for four DBMS products (two open-source and two closed-development). We ask questions about the potential effectiveness of *design diversity* – deploying two different products. Fault reports are the only publicly available dependability evidence for these products, so our study concerns *fault diversity* among them. Complete failure logs would be much more useful as statistical evidence, but they are not available. Many of the vendors discourage users from reporting already known bugs, and detailed failure data are rarely available even to the software vendors themselves.

In a first study [9], we looked at the set of bugs reported for one release of each DBMS product. For each bug, we took the bug script (a sequence of SQL statements) that would trigger it and ran it on all four DBMS products (if possible), to check for coincident failures: if the bug script does not trigger failures in the other DBMS product, there is

³ By “self-evident failures” we will mean failures that a generic client of the DBMS product can detect without depending on knowledge of the specific database and its semantics. They are those failures that – as seen by the client – consist in issuing an error message to the client, spontaneously aborting a transaction, “hanging” or crashing.

evidence that software fault tolerance would tolerate that fault. We found that a high number of reported faults would not be tolerated (or even detected) by existing, non-diverse fault-tolerant schemes but did not cause coincident failures in any two DBMS products, offering a way of tolerating them.

These intriguing results suggested a potential for considerable dependability gains from using diverse OTS DBMS products, but they only concerned a *specific snapshot* in the evolution of these products. We therefore ran a follow-up study with later releases of DBMS products (thus with different set of bug reports), with results that substantially confirm the previous ones. This paper reports the complete results of the two studies.

The rest of the paper is organized as follows: in Section 2, we briefly discuss the architectural issues in software fault tolerance with DBMS products – feasibility, design alternatives and performance issues – since they determine the usefulness of the empirical results we report; Section 3 presents the results of the two empirical studies of known bugs of DBMS products, including the comparisons between older and newer releases of two DBMS products; Section 4 contains a discussion of the implications of our studies; Section 5 contains a review of related work on database replication, interoperability of databases, empirical evidence on DBMS products’ faults and failures and diversity with off-the-shelf components and Section 6 contains conclusions and outlines of further work.

2 ARCHITECTURAL CONSIDERATIONS

2.1 Current Solutions for DBMS Replication

Standard solutions for automatic fault tolerance in databases use the mechanisms of atomic transactions and/or checkpointing to support backward recovery of failed computations, which can be followed by retry of the failed statements⁴. These solutions will tolerate transient faults, if detected, and if combined with replication will mask permanent faults, without service interruption.

Various data replication solutions exist [10], [11], [12], [13], [14]. In commercial DBMS products, they are often called “fail-over” solutions: following a (crash) failure of the primary DBMS product, the load is transparently taken over by a separate installation of the DBMS product holding a redundant copy of the database, at the cost of aborting the transactions affected by the crash. Multiple copies may be used. The code for fault tolerance is integrated inside the DBMS product. A recent survey [15] calls this a “white box” solution. As an alternative, replication can be managed by middleware separate from the DBMS products: “black box” solutions (fault tolerance is entirely the responsibility of the middleware), or “grey box” (the middleware exploits useful functions available from the DBMS products [16]). Our discussion here will refer to “black box” solutions: the only ones that can be built without access to OTS source code, and most convenient for studying the design issues in the use of redundancy and diversity. We will assume that fault

⁴ We will use the term “statement” to refer to the SQL requests that are sent to the server. These may be read or write *data manipulation language* (DML) statements or *data definition language* (DDL) statements.

tolerance is managed by a layer of middleware; clients see the fault-tolerant database server via this middleware layer, which co-ordinates the redundant channels.

Existing data replication solutions use sophisticated schemes for reducing the overhead involved in keeping the copies up to date. Their common weakness is their dependence on the assumption of “fail-stop” or at least “self-evident” failures. This assumption simplifies the protocols for data replication, and allows some performance optimization. For instance, in the Read Once Write All Available (ROWAA) [10] replication protocol the read statements are executed by a single replica while the write statements are executed by all replicas. These fault-tolerant solutions are considered adequate by standardizing bodies [17], despite the assumption being false in principle. Some recent solutions [18] seek further optimization by executing the write statements on a single replica, which then propagates the changes to all the (available) replicas.

As we shall see, current OTS DBMS products suffer from many bugs that cause non-crash, non self-evident failures. The failures that these cause may be undetected erroneous responses to read statements, and/or incorrect writes to all the replicas of the database.

For these kinds of failure, the current data replication solutions are deficient, in the first place from the viewpoint of error detection. Two kinds of remedy are possible:

- *database-*, or *client-specific* solutions that depend on the client (an automatic process or a human operator) to run reasonableness checks on the outputs of the DBMS product and order recovery actions if it detects errors. Good error detection may be achieved by exploiting knowledge of the semantics of the data stored and the processes that update them. This knowledge may also support more efficient error recovery than simple rollback and retry. The main disadvantages are high implementation cost (especially with a workforce generally unaware of the need for fault tolerance), high run-time cost, at least for human-run checks, and the possibility of low error detection coverage if the database is – as common – the sole repository of the data⁵.
- *generic* solutions that use active replication [19] for error detection, so that errors can be detected by comparing the results produced by redundant executions, and/or corrected, via voting or copying the results of correct executions.

2.2 Diversity

Replication will give a basis for effective fault tolerance if the multiple copies of the database do not usually fail together on the same demand, or at least they tend not to fail with identical erroneous results. To pursue such *failure diversity*, a designer can use various forms of diversity in a redundant system:

- simple separation of redundant executions. This is the weakest form, but it may yet tolerate some failures. It is well known that many bugs in complex, mature software

⁵ Simple reasonableness or “safety” checks are often available, but have limited efficacy against some failure scenarios. E.g., reasonableness checks may prevent the posting of incredibly large movements in a company’s accounts, yet allow many small systematic errors, allowing large cumulative errors to build up before the problem comes to light.

products are “Heisenbugs”⁶ [20], i.e., they cause apparently non-deterministic failures. When a database fails, its identical copy may not fail, even with the same sequence of inputs. Even repeating the same operations on the same copy of a database after rollback may in principle not replicate the same failure;

- *design diversity*, the typical form of parallel redundancy for fault tolerance against design faults: the multiple replicas of the database are managed by diverse DBMS products;
- *data diversity* [21]: thanks to the redundancy in the SQL language, a sequence of one or more SQL statements can be “rephrased” into a different but logically equivalent sequence to produce redundant executions (on multiple DBMS products or on a single one), thus giving a better chance of a failure not being repeated when the rephrased sequence is executed on another replica of even the same DBMS product. Two of the present authors have reported elsewhere [22] on a set of “rephrasing rules” that would tolerate at least 60% of the bugs examined in our studies.
- *configuration diversity* (which can be seen as a special form of data diversity). DBMS products have many configuration parameters, affecting e.g. the amount of system resources they can use (amount of RAM and/or the “page size” used by the database), or the degree of optimization to be applied to certain operations: given the same database contents, varying these parameters between two installations can produce different implementations of the data and the operation sequences on them, and thus decrease the risk of the same bug being triggered in two installations of the same DBMS product by the same sequence of SQL statements. Another example is the use of “hints” with SQL statements: they are directives to the “query optimizer”⁷ to change the execution plan of the SQL statements.

These precautions can in principle be combined (for instance, data diversity can be used with diverse DBMS products), and implemented in various ways, including manual application by a human operator.

Among the above forms of diversity, design diversity appears the most likely to avoid coincident failures in redundant executions, but it may impose substantial limitations or design costs. In the first place, OTS DBMS products, even if they nominally implement the operations of the standard SQL language, in practice use different “dialects”: they use different syntax for commands that are semantically the same (this problem can be solved via automatic, on-the-fly translation); more importantly, each offers extra, non-standard features, which would require either more complex translation⁸, and/or clients to be limited to

⁶ A term was introduced by Gray [20], defining two types of bugs: “Bohrbugs” appear to be deterministic (the failures they cause are easy to reproduce in testing); “Heisenbugs”, are difficult to reproduce as they only cause failures under special conditions: “strange hardware conditions (rare or transient device fault), limit conditions (out of storage, counter overflow, lost interrupt, etc.) or race conditions”.

⁷ A component of a DBMS product that attempts to determine the most efficient way to execute a statement

⁸ “Rephrasing” of statements, mentioned above as a form of data diversity, can also be useful to overcome this difficulty: a statement sent by a client using the dialect of one DBMS product may not “make sense” to another DBMS product; but a logically equivalent, rephrased version of the

using a common subset among the features of the diverse DBMS products. In addition, many aspects of database operation are specified in a non-deterministic fashion, making the goal of ensuring consistency among replicas difficult even with same-product replication, and more so with diverse replication.

A special case of design diversity is using successive releases of the same DBMS product. This will avoid or greatly reduce the problems due to “dialect” differences. It may be expected to tolerate fewer faults, since the successive releases will share large portions of their code, including some bugs; but they may be attractive for “smoothing out” upgrades which may otherwise cause peaks of unreliability in a database installation, due to the new faults introduced, and at the same time evaluating the new release to decide when it has reached sufficient dependability to be used alone in the installation. Similar practices have been applied for embedded and safety critical systems [24], [25].

We now discuss briefly the architectural options available in designing automated fault tolerance solutions with some form of diversity applied to OTS DBMS products. A basic “black box” replication architecture delegates the management of redundancy to a layer of middleware [26], as in Fig. 1, so that the multiple DBMS products appear to clients as a single server. There may be any number of channels, though typical values would be one (using “time redundancy” – repeating the execution on the single DBMS product – when needed), two or three (the minimum that allows error masking through voting). We will normally refer to systems with two replicas, unless otherwise noted.

This basic architecture can be used for various fault tolerance strategies, with different trade-offs between coverage for various types of failures, performance, ease of integration etc [27]. The most serious design trade-offs with all these schemes concern ensuring replica determinism, for replication schemes that require it. The design difficulty is that each DBMS product has its own concurrency control strategy, and these are non-deterministic and may be different between products. Proprietary replication solutions

can deal with this problem by using knowledge of the implementation of the DBMS product. For a middleware layer dealing with generic OTS products, this is more difficult, especially since commercial vendors may keep these details secret. The middleware can instead artificially serialize statements in the same way on all replicas [28], [29]. This solution carries performance costs, but these will be acceptable for many installations, though intolerable on others, depending on the amount and pattern of write transactions in a specific installation.

A separate requirement, easier to satisfy, is that any voting/comparison algorithm need to allow for “cosmetic” differences between equivalent correct results issued by different DBMS products, e.g. differences in the padding blank characters in character strings or different numbers of digits in the representations of floating point numbers. Trade-offs exist here between embedding in the algorithm more knowledge about the idiosyncrasies of each specific product, and keeping it more generic at the cost of possibly lower coverage.

2.3 Design Options for Fault Tolerance via Diverse Replication

2.3.1 Detecting Server Failures

Erroneous responses to read statements can be detected by comparing the outputs of the channels, detecting those non-self-evident failures that cause some discrepancy between these outputs.

Both design diversity and data diversity increase the chance of detection, compared to simple replication. Replica determinism is necessary, i.e. discrepancies between correct results must be rare as they may cause correct results to be flagged as erroneous, and thus a performance penalty. Self-evident DBMS product failures are detected as in a non-diverse DBMS products, via server error messages (i.e. via the existing error detection mechanisms inside the DBMS products) and time-outs.

Erroneous updates to the databases that will only cause

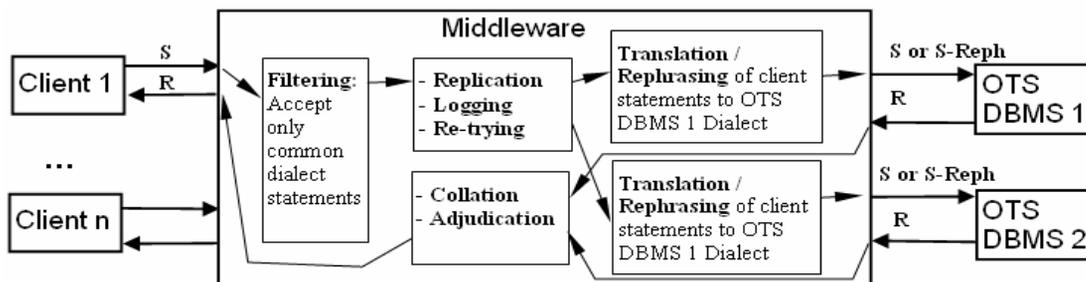


Fig. 1 - A stylized design of a fault-tolerant database server with two *channels*. Each channel includes an installation of an OTS DBMS product (these may be the same or different products, including different releases of a same product) and a replica of the database. The *middleware* must ensure connectivity between the *clients* and the DBMS products, some filtering of the statements sent by clients (e.g. returning error messages to the client for statements that are not supported by both the underlying OTS DBMS products), replication and concurrency control, management of fault tolerance (error detection; error containment, diagnosis and correction; state recovery), as well as *translation* of SQL statements (“S” in the figure) sent by the clients to the dialects of the respective OTS DBMS products (translation may be done in off-the-shelf add-on components). Support for “data diversity” through “*rephrasing*” may also form part of the same components which perform translation: *rephrasing* rules will produce rephrased versions – “S-reph” in the figure – of the statements sent by clients. The middleware must also *adjudicate* the results – “R” in the figure – from the OTS DBMS products and return a result to the client[s].

statement might be supported by both products. E.g. the TRUNCATE command is a specific feature of the PostgreSQL DBMS (and is faulty in release 7.0; see bug report 20 [23] for details); but all uses of TRUNCATE can be translated into uses of the DELETE command, which is implemented in all the SQL compliant DBMSs.

output discrepancies in the future are also a concern. To detect them, the middleware can compare the contents of the database replicas, via the standard read commands of the DBMS products. There is a degree of freedom in how

much should be compared, allowing latency/performance trade-offs. The middleware can just ask each DBMS product for the list of the records modified in each write operation, and then read and compare their contents. In principle, though, a buggy DBMS product could modify other records as well, and omit them from the list it returns. So, a designer could decide to compare a superset of the data that appear to be affected, trading off time for better error detection.

Error detection can be run in a more or less pessimistic mode: in the most pessimistic mode, at each operation the middleware performs all its comparisons before forwarding to the client the response from the DBMS product[s]. More optimistically, it can forward most responses immediately, and run the checks in parallel with the subsequent operation of the client and DBMS products. A natural synchronization point is at transaction commit: the middleware only allows the transaction to commit if it detected no failures. Thus the designer can trade off the error latency against the overhead imposed by the fault-tolerant operation.

In addition, the middleware can use slack capacity for a background audit task, comparing the complete contents of the database replicas.

2.3.2 Error Containment, Diagnosis and Correction

Error containment is tightly linked to detection. For read statements, the middleware receives multiple responses for each statement sent to the diverse channels, one from each of them, and must return a single response to the client. In general, the middleware will present to the client a DBMS product failure as a correct but possibly delayed response (masking), or as a self-evident failure (crash - the behavior of a "fail-silent" fault-tolerant server; or an error message - a "self-checking" server). DBMS product failures can be masked to the clients, if the middleware can select a result that has a high enough probability of being correct:

- if more than two redundant responses are available, majority voting can be used to choose a consensus result, and to identify the failed replica which may need a recovery action to correct its state.
- with only two redundant channels, if they give different results the middleware cannot decide which one is in error. A possibility is not to offer masking, but simply a clean failure to be followed by manual diagnosis of the problem. Alternatively, additional redundant execution can be run by replaying the statements, possibly with "data diversity", i.e., rephrasing the statements [22].

Depending on how redundant executions are organized, the middleware may need to resolve rather complex scenarios, e.g., two diverse DBMS products, A and B, may give different responses upon first submission for a read statement, while upon resubmission of a rephrased statement A produces an error message and B a result matching A's previous result; but this is a standard adjudication problem [30], [31], [32] for which the design options and trade-offs are well known.

Again, the need for replica determinism is the main design issue with these schemes.

2.3.3 State Recovery

Besides selecting probably correct results, adjudication will identify probably failed DBMS products (diagnosis). This improves availability: the middleware can selectively direct recovery actions at the DBMS product diagnosed as having failed, while letting the other DBMS product(s) continue providing the service.

The state of a replica DBMS product can be seen as composed of the state of permanent data in the database and that of volatile data in the DBMS product's variables. For erroneous states of the latter, since the middleware cannot see the internal state of each executing DBMS product, some form of "rejuvenation" [33] must be applied, e.g. stopping and restarting the DBMS product.

As for state recovery of the database contents, it can be obtained:

- via standard backward error recovery - rollback followed by retry of logged write statements -, which will sometimes be effective (failures due to Heisenbugs), at least if the failures did not violate the ACID property in the affected transactions. "Data diversity" will extend the set of failures that can be recovered this way. To command backward error recovery, the middleware can use the standard database transaction mechanisms: aborting the failed transaction and replaying its statements may produce a correct execution. Alternatively or additionally, it can use checkpointing [34]: the middleware orders the states of the database replicas to be saved at regular intervals (by database "backup" commands: e.g., in PostgreSQL the `pg_dump` command). After a failure, a database replica is restored to its last checkpointed state and the middleware replays the sequence of (all or just write) statements since then (the redo log provided in some DBMS products cannot be used because it might contain erroneous writes). Finer granularity of recovery can be achieved by using the checkpoint-rollback mechanism within transactions: this allows the handling of exceptions within transactions, and should be applied when using data diversity through "rephrasing" [22];
- additionally, diversity allows one to achieve forward recovery by essentially copying the state of a correct database replica into the failed one (similarly to [35]). Since the formats of database files differ between the DBMS products, the middleware would need to query the correct channel[s] for their database contents and command the failed channel to write them into the corresponding records in its database, similar to the solution proposed in [36]. This would be expensive, perhaps to be completed off-line, but a designer can use multi-level recovery, in which the first step is to correct only those records that have been found erroneous on read statements.

3 OUR STUDIES OF BUG REPORTS FOR OFF-THE-SHELF DBMS PRODUCTS

3.1 Generalities

We use the following terminology. The known bugs for the OTS DBMS products are documented in bug report repositories (i.e. bug databases, mailing lists etc). Each *bug report*

contains a description of the bug and a *bug script* for reproducing the *failure* (the erroneous behavior that the reporter of the bug observed). The bug script may come with indications on the database states that are preconditions for the failure (e.g., in the form of statements to issue for the database to reach one such state), plus the statements (and values for their parameters) which reproduce the *failure*. In our study we collected these bug reports and *ran* the bug scripts on installations of each of the DBMS products we used (we will use the phrase “*running a bug*” for the sake of brevity).

What constitutes an individual bug is of course not definable by any a priori rule [37, section 2.2]: people characterize a bug in terms of the apparent mistakes made by the developers, of code changes required to fix it, and/or of circumstances on which the software fails. We define a “demand” as the complete circumstances (i.e. an initial state plus a series of statements) that would cause failure. A bug report does not necessarily identify the whole set of demands (the “failure region”) on which the product fails and would no longer fail if the bug were corrected. When running a bug script, we usually tested all DBMS products on at least one demand (the same for all DBMS products in our study) mentioned in the bug report, and we listed the bug as present in all DBMS products where the demand caused a failure. In some cases, we also tested the DBMS products with other similar demands - variations of the statements and/or parameters specified in a bug script. We did this when a bug script did not seem to trigger a failure in the DBMS product to which it related, to check whether the bug did appear to be present, but the reporter may have been imprecise in characterizing the conditions for triggering it; and when a bug script caused failures in more than one DBMS products, to study and compare the “failure regions” identified in the two products, especially to determine whether they coincide and whether the DBMS products fail identically throughout them.

3.1.1 Reproducibility of Failures

As mentioned earlier, DBMS products offer features that extend the SQL standard, and these extensions differ between products. Bugs affecting these extensions literally cannot exist in a DBMS product that lacks them. We called these bugs “dialect-specific”. For example, Interbase bug 217138 [23] affects the use of the UNION operator in VIEWS, which PostgreSQL 7.0 VIEWS do not offer, and thus cannot be run in PostgreSQL 7.0: it is a dialect-specific bug.

Another reproducibility issue arises when a bug script does not cause failure in the DBMS product for which the bug was reported. We called these bugs ‘Unreproduced’ bugs. These bugs may be Heisenbugs [20] or bugs reported without enough detail for reproducing them. As we mentioned, for these bugs we attempted to run more variations of the incomplete bug script described in the bug reports. In some cases, more complete bug scripts were also posted after our collection period or they were reported in a different mailing list other than the main bugs’ repository of the respective DBMS product. This allowed us to trigger a few of the bugs that we had previously considered to be ‘Unreproduced’ (therefore the statistics we report here differ slightly from the preliminary results we reported in [9]).

3.1.2 Classifications of Failures

We ran each bug first on the DBMS product for which it was reported, and then (after translating the script into the appropriate SQL dialect[s]) on the other DBMS product[s]. We classified bugs into Reproduced and Unreproduced and into dialect-specific and non-dialect-specific bugs, as explained previously, and failures into different categories that would require different fault tolerance mechanisms:

Engine Crash failures: a crash or halt of the core engine of the DBMS product.

Incorrect Result failures, which are not engine crashes but produce incorrect outputs: the results do not conform to the DBMS product’s specification or to the SQL standard.

Performance-related failures. We classified as performance failures: i) failures that are so classified in bug reports; ii) failures observed by us if either the DBMS product clearly “hung” or, whatever the observed latency, the bug script generated a query plan indicating potential performance problems, e.g. with an un-utilized column “index” in a SELECT statement using that column.

Other failures: e.g. security related failures, such as incorrect privileges for database objects (tables, views etc.)

We further classified failures according to their detectability by a client of the DBMS products. We already came across some of these terms in section 2. In the context of our study and with reference to the types of failures defined above then the following definitions apply:

Self-Evident Failure: engine crash failures, internal failures signaled by DBMS product exceptions (error messages) or performance failures

Non-Self-Evident Failures: incorrect result failures without DBMS product exceptions, with acceptable response time.

For clients with access to at least two diverse DBMS products the failures would be:

Divergent failures: any failures where DBMS products return different results. All failures affecting only one out of two (or at most n-1 out of n) DBMS products are divergent. Even if all fail but ‘differently’ the failure will still be divergent.

Non-divergent failures: the ones for which two (or more) DBMS products fail with identical symptoms. For some bugs, all demands we ran caused non-divergent failures, for others only some demands did. In the tables that follow we use the labels “non-divergent – all demands” and “non-divergent – some demands” for these two cases.

All the *divergent* or *self-evident* failures are *detectable* by a client of the DBMS products when at least two replicas of the database are available, on different DBMS products. The remaining failures (*non-divergent* and *non-self-evident*) are *non-detectable*.

3.2 The First Study

3.2.1 Bug Reports

In our first study [9] we used a total of four DBMS products: two commercial (Oracle 8.0.5 and Microsoft SQL Server 7, without any service packs applied) and two open-source ones (PostgreSQL Version 7.0.0 and Interbase Version 6.0). Interbase, Oracle and MSSQL were all run on the Windows 2000 Professional operating system; PostgreSQL

7.0.0 (not available for Windows) was run on RedHat Linux 6.0 (Hedwig). For the sake of brevity, we will use the following abbreviations:

- PG 7.0⁹ - for PostgreSQL 7.0.0
- IB - for Interbase 6.0
- OR - for Oracle 8.0.5
- MS - for Microsoft SQL Server 7

For each of these DBMS products there is an accessible repository of reports of known bugs. We collected the IB bugs from SourceForge [38], the PG 7.0 bugs from its mailing list, [39], MS bugs from its service packs site [40] and OR bugs from the Oracle Metalink [41].

We only used bugs that caused failure of a DBMS product's core engine. Other bugs, e.g. causing failures of a client application tool, various connectivity (JDBC/ODBC etc.) or installation-specific bugs were not included in the study, because these functions in a future fault-tolerant architecture would be provided by the middleware.

For each reported bug, we attempted to run the corresponding bug script. Full details are available in [23] (and also provided as Supplement A).

3.2.2 Detailed Results

In total, we included in the study 181 bug reports: 55 for IB, 57 for PG, 51 for MS and 18 for OR. Out of these 181 bugs, 70 were dialect-specific (could be run in only one of the four DBMS products); 58 could be run in all four DBMS products; 26 could be run in only two DBMS products and 27 in only three DBMS products. Each bug report was unique, i.e., none of the bugs were reported for more than one DBMS product.

Table 1 contains the results of the first study. The structure of the table is as follows. Each grey column lists the

deduced). The 47 bugs that caused failures are further classified in the part of the column below the double horizontal line, after the "Failure observed" row. All the performance failures and all the DBMS product engine crashes are self-evident. Incorrect Result failures and "Other" failures can be self-evident or non-self-evident, depending on whether the DBMS product gives an error message.

To the right of the grey column, three columns present the results of running the IB bugs on the other three DBMS products. For example, we can see that 24 of the IB bugs cannot be run in PG 7.0 (dialect-specific bugs). Out of 55 Interbase bugs we managed to run 31 in PG 7.0; only one caused a failure in both IB and PG 7.0. This particular failure was a non-self-evident incorrect result, as can be seen from the table. Details about the bugs causing coincident failures were given in [9]. Three bugs are classified as "Undecided Performance": this means that the bug report indicated a "performance failure" but we could not decide, by analyzing the query plan and observed response time, whether a performance failure also occurs in PG 7.0.

As for the failure types, we can see that most of the bugs for each DBMS product cause Incorrect Result failures. The percentage of non-self-evident failures is also high: they range from 44% for MS to 66% for IB. Engine crashes are less frequent: they range from 13% for MS to 21.5% for OR.

3.2.3 Implications for Fault Tolerance: Two-Version Combinations

We now look more closely at the two-version combinations of the four different DBMS products in our study. We want first to find out how many of the coincident failures are *detectable* (i.e. *divergent* or *self-evident*) in the two-version systems. Table 2 contains a summary of the results on each

TABLE 1. STUDY 1: RESULTS OF RUNNING THE BUG SCRIPTS ON ALL FOUR DBMS PRODUCTS. ABBREVIATIONS: IB – INTERBASE 6.0; PG 7.0 - POSTGRESQL 7.0.0; OR – ORACLE 8.0.5; MS – MICROSOFT SQL SERVER 7.

	IB	PG 7.0	OR	MS	PG 7.0	IB	OR	MS	OR	IB	MS	PG 7.0	MS	IB	OR	PG 7.0	
Total bug scripts	55	55	55	55	57	57	57	57	18	18	18	18	51	51	51	51	
Bug script cannot be run (Functionality Missing)	n/a	24	21	17	n/a	33	27	24	n/a	14	14	13	n/a	36	35	31	
Total bug scripts run	55	31	34	38	57	24	30	33	18	4	4	5	51	15	16	20	
Undecided performance	0	3	3	3	0	0	0	0	0	0	0	1	0	3	4	2	
No failure observed	8	27	31	33	5	24	30	31	4	4	4	3	12	11	12	12	
Failure observed	47	<u>1</u>	0	<u>2</u>	52	0	0	<u>2</u>	14	0	0	<u>1</u>	39	<u>1</u>	0	<u>6</u>	
Types of failures	Poor Performance	3	0	0	0	0	0	0	1	0	0	0	6	0	0	0	
	Engine Crash	7	0	0	0	11	0	0	3	0	0	0	5	0	0	0	
	Incorrect Result	Self-evident	4	0	0	<u>1</u>	14	0	0	<u>1</u>	3	0	0	10	0	0	<u>6</u>
		Non-self-evident	23	<u>1</u>	0	<u>1</u>	20	0	0	<u>1</u>	7	0	0	<u>1</u>	17	<u>1</u>	0
	Other	Self-evident	2	0	0	0	2	0	0	0	0	0	0	1	0	0	0
Non-self-evident		8	0	0	0	5	0	0	0	0	0	0	0	0	0	0	

results produced when the bugs reported for a certain DBMS product were run on that DBMS product. For example, we collected 55 known IB bugs, of which, when run on our installation of IB, 8 did not cause failures (Unrepro-

duced) of the six possible two-version combinations¹⁰.

⁹ For PostgreSQL we also use the release number in the identifier since we will report later on results of one of its later releases.

¹⁰ Here we only include bugs (reported for any of the four DBMS products) that could be run on both DBMS products, i.e. we exclude dialect-specific bugs. For instance, Table 2 shows that there were a total of 71 bugs that could be run on both IB and PG 7.0. In detail, 31 of these were reported for IB and 24 for PG; these two numbers can be deduced from Table 1. The remaining 16 were bugs of either OR or MS which could be run on both IB and PG 7.0 - these numbers are not directly deducible from Table 1 due to

TABLE 2. STUDY 1: SUMMARY OF RESULTS FOR THE TWO-VERSION COMBINATIONS.
 ABBREVIATIONS: S.E. — SELF-EVIDENT FAILURE; N.S.E. - NON-SELF-EVIDENT FAILURE.

Pairs of DBMS Products	Total number of bug scripts run	Bugs scripts causing failure (in at least one DBMS product)	One out of two DBMS products failing		Both DBMS products failing						
			s.e.	n.s.e.	Non – Divergent				Divergent		
					All Demands		Some Demands		1 s.e. & 1 n.s.e.	2 s.e.	2 n.s.e.
					s.e.	n.s.e.	s.e.	n.s.e.			
IB + PG 7.0	71	49	22	26	0	<u>1</u>	0	0	0	0	0
IB + OR	69	32	11	21	0	0	0	0	0	0	0
IB + MS	78	43	17	23	<u>1</u>	<u>2</u>	0	0	0	0	0
PG 7.0 + OR	72	33	16	16	0	0	0	0	0	0	<u>1</u>
PG 7.0 + MS	85	48	20	21	0	<u>1</u>	0	0	<u>3</u>	<u>3</u>	0
OR + MS	80	18	11	7	0	0	0	0	0	0	0

Only twelve coincident failures occurred (note that there were thirteen bugs that caused failures in a different DBMS product than the one for which they were reported (as detailed in Table 1); one bug (MS bug report 56775) [23], although reported for MS, did not cause failure in MS (Unreproduced) but did cause failure in PG 7.0); only four of these twelve are non-detectable. We can see that diversity allows detection of failures for at least 95% of these bugs (41 out of 43, for the IB+MS pair). Moreover, it would support masking and forward recovery (following the self-evident failure of a single channel) for a fraction of bugs varying between 11/32 (34%) for the IB+OR pair) and 11/18 (61%) for the OR+MS pair. More details on these bugs are in [9] and [23].

3.3 The Second Study

3.3.1 Description of the Study

To repeat the study on later releases of DBMS products, we collected 92 new bug reports for the later releases of the open-source DBMS products: PostgreSQL 7.2 and Firebird 1.0¹¹ (abbreviated as PG 7.2 and FB respectively). We excluded the closed-development DBMS products as the bug scripts needed to trigger the faults were missing in most of their bug reports. But we still translated the new bug scripts of bugs reported for the open-source DBMS products into the dialects of the closed-development ones, and ran them in the same releases that we used in the first study (Oracle 8.0.5 and MSSQL 7.0). The results of the second study are shown in Table 3 (for full details see [23]). The classifications of faults and failures are as defined in section 3.1.

Incorrect results are still the most frequent failures. Engine crashes are slightly more frequent than in the first study but still no more than 22.2%. The number of non-self-evident failures is lower than in the first study: 35% for PG 7.2 and 53% in FB. The number of bugs causing coincident failures was again low: in the second study we observed a total of 5 coincident failures. None of the bugs caused failures in more than two DBMS products. Full details of the coincident failures will be given in section 3.3.3.

some bugs being dialect-specific for one DBMS product but not another; they can however be obtained from [23, Table 3]).

¹¹ Firebird is the open-source descendant of Interbase 6.0. The later releases of Interbase are issued as closed-development by Borland.

3.3.2 Implications for Fault Tolerance: Two-Version Combinations

Table 4 shows the results of the two-version combinations of the 4 DBMS products used in the second study. None of the bugs caused non-detectable failures for all demands. One bug caused a non-detectable failure only for some demands, but is detectable for vast majority of others. Three bugs caused self-evident failures in both DBMS products and one caused non-self-evident failure in one and self-evident failure in the other.

So, diversity allows detection of failures for all these bugs. It would allow masking and forward recovery (following the self-evident failure of a single channel) for a fraction of bugs varying between 11/28 (39%) for the FB+MS pair and 12/20 (60%) for the PG 7.2+MS pair.

3.3.3 Common Bugs

It is interesting to describe in some more detail some of the bugs that caused coincident failures, listed in Table 5¹², and speculate about the probable frequency and severity of the failure observed.

Arithmetic-related bugs

Firebird bug 926001 [23] causes non-self-evident failure in both FB and PG 7.2 when the DBMS product is asked to add two values of type Timestamp (a timestamp value contains both date and time information). Due to rounding errors, FB always gives a result that is 1 second less than the correct result, whereas PG 7.2 adds the dates but not the time of the second timestamp value (i.e. it treats the operation as $\text{Timestamp}_1 + \text{Date}_2$). The failure rate for this bug would be highest in applications that require high precision arithmetic computations with timestamp datatypes. On most demands the erroneous results of the two DBMS products would be different: the failure is non-divergent only for some (probably rare) demands.

FB bug 926624 [23] causes a crash in both FB and MS. The crash is due to a stack overflow from attempting to use in the column part of the SELECT statement an arithmetic expression longer than: 8000 characters in FB; 2834 characters in MS. Therefore FB fails for a smaller set of demands than MS. The expected correct behavior is for the DBMS product to process the statements, or to give an error message that warns the user of the maximum limit for an ex-

¹² Similar accounts for bugs in Study 1 are in our preliminary report [9].

TABLE 3. STUDY 2: RESULTS OF RUNNING THE BUG SCRIPTS OF FB AND PG ON ALL FOUR DBMS PRODUCTS.
ABBREVIATIONS: FB – FIREBIRD 1.0; PG 7.2 - POSTGRESQL 7.2; OR – ORACLE 8.0.5; MS – MICROSOFT SQL SERVER 7.

		FB	PG 7.2	OR	MS	PG 7.2	FB	OR	MS	
Total bug scripts		43	43	43	43	49	49	49	49	
Bug script cannot be run (Functionality Missing)		n/a	12	15	13	n/a	29	29	30	
Total bug scripts run		43	31	28	30	49	20	20	19	
Undecided performance		0	1	2	1	0	2	2	0	
No failure observed		4	29	26	27	4	17	18	18	
Failure observed		39	<u>1</u>	0	<u>2</u>	45	<u>1</u>	0	<u>1</u>	
Types of failures	Poor Performance	4	0	0	0	5	0	0	0	
	Engine Crash	6	0	0	<u>1</u>	10	0	0	<u>1</u>	
	Incorrect Result	Self-evident	7	0	0	0	13	<u>1</u>	0	0
		Non-self-evident	20	<u>1</u>	0	<u>1</u>	15	0	0	0
	Other	Self-evident	1	0	0	0	1	0	0	0
		Non-self-evident	1	0	0	0	1	0	0	0

TABLE 4. STUDY 2: SUMMARY OF RESULTS FOR THE TWO-VERSION COMBINATIONS.
ABBREVIATIONS: S.E. – SELF-EVIDENT FAILURE; N.S.E. - NON-SELF-EVIDENT FAILURE.

Pairs of DBMS Products	Total number of bug scripts run	Bugs scripts causing failure (in at least one DBMS product)	One out of two DBMS products failing		Both DBMS products failing						
			s.e.	n.s.e.	Non – Divergent				Divergent		
					All Demands		Some Demands		1 s.e. & 1 n.s.e.	2 s.e.	2 n.s.e.
					s.e.	n.s.e.	s.e.	n.s.e.			
FB + PG 7.2	51	47	26	19	<u>1</u>	0	0	<u>1</u>	0	<u>0</u>	0
FB + OR	46	25	10	15	0	0	0	0	0	0	0
FB + MS	46	28	11	15	0	0	<u>1</u>	0	<u>1</u>	0	0
PG 7.2 + OR	47	21	13	8	0	0	0	0	0	0	0
PG 7.2 + MS	47	20	12	7	0	0	<u>1</u>	0	0	0	0

TABLE 5. BUGS THAT CAUSE COINCIDENT FAILURES

On which additional DBMS product was failure observed?				
	FB	PG	OR	MS
FB	N/A	<u>1</u> - (Bug ID 926001)	0	<u>2</u> - (BugIDs 910423, 926624)
PG	<u>1</u> (Bug report date 16/05/2003)	N/A	0	<u>1</u> - (BugID 847)

For which DBMS product was the bug reported?

pression length. The failure rate for this bug would probably be low for most installations, as SELECT statements would seldom contain such long arithmetic expressions.

Miscellaneous bugs

FB bug 910423 [23] causes failure in both FB and MS. Fig. 2 shows the demands for which they fail. The failure consists in allowing the datatype of a table column to be changed from integer to string even when the string type is specified to be shorter than needed to hold the data already stored in the column. The expected correct behavior is for the DBMS product to refuse (with an error message) to change the datatype of either any column that already contains data, or at least those containing data that wouldn't fit in the new length specified. If a client later tries to read the column affected, the two DBMS products react differently: FB responds with an error message (self-evident failure), while MS returns a '*' symbol. We have therefore classified the failure as divergent. As shown, MS actually fails on a superset of the demands on which FB does. It is difficult to conjecture how often applications change the datatypes of

columns and hence the likely failure rates for this bug. The severity of this failure is different in the two DBMS products. FB does not lose the data stored in the column: if you just change the type again to a long enough string (≥ 10 in the example above) then the data can again be read. MS instead truncates the data item to the new length set, so that it is irremediably lost.

PG 7.2 bug 847 [23] causes failure in both PG 7.2 and MS. PG 7.2 allows the creation of exceptions that return a message longer than 4000 characters, but then crashes if the exception is raised. The correct behavior is for a DBMS product to give an error message once its maximum length for exception messages is reached: either at exception definition or when attempting to raise the exception. The same problem occurs in MS, but the threshold message length is even smaller (440 characters), and thus failures would be more frequent.

The PG 7.2 bug reported on 16 May 2003 (with no ID in the PG 7.2 mailing list [23]) activates an error message in PG 7.2 and FB, although no error exists. The bug script is:

```
CREATE TABLE TEST2 (V1 INT, V2 INT, CONSTRAINT
UQ_TEST UNIQUE (V1,V2));
INSERT INTO TEST2 VALUES (0,0);
INSERT INTO TEST2 VALUES (0,1);
INSERT INTO TEST2 VALUES (0,2);
UPDATE TEST2 SET V2=V2+2;
Violation of PRIMARY or UNIQUE KEY constraint
"UQ_TEST" on table "TEST2"
```

The UPDATE statement in this script changes the contents of the database during intermediate steps so as to vio-

late the `UNIQUE CONSTRAINT`¹³ although the final state does not violate it. OR and MS correctly execute the script without error messages; PG 7.2 and FB do the `UNIQUE CONSTRAINT` checks at intermediate states (in this case after each row is updated), which causes the exception to be raised. The failure is not specific to this bug script. It can be triggered with any `UNIQUE CONSTRAINT` on integer, real or float datatypes affecting multiple columns, whenever an update is attempted that will (at an intermediate step during the execution) set a value of a row to that of an existing row in the table, although at the end of the execution of the statement no violations would be present. On every set of parameter values that we tried, either both DBMS products failed or neither did. The failure rate for this bug is expected to be relatively high in update-intensive applications if `UNIQUE CONSTRAINT` is used on multiple columns.

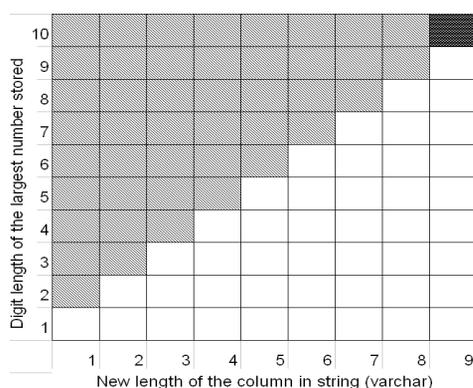


Fig. 2. FB bug 910423: demands on which MS fails (light grey shaded boxes) and demands for which both FB and MS fail (dark grey).

3.4 Newer vs. Older Releases (Open-Source DBMS Products)

We now look more closely at those DBMS products that were used in both studies (i.e. the two open-source products). We ran all the new bugs reported for the newer releases on the older releases, to check how many already existed there. The results are in the leftmost eight columns of Table 6 (full details are in [23]).

The structure of the table is the same as that of Table 1 and Table 3. We can see that 33 bugs reported for FB also cause failure in the older release IB. Of the six that do not cause failures in IB, four were Unreproduced in FB. So only 2 bugs that caused failure in FB (the new release) appear to be new bugs, introduced in functionalities that used to work correctly. The reason for the little difference that we see in running the bugs of FB in IB could be that FB 1.0 was mainly a bug fix release, with no major enhancements, which probably also reduced the number of potential new problems that could be introduced.

The situation is different for PG 7.2, which featured many more enhancements, for example the support for OUTER JOINS in SELECT statements. We can see that 13 of the bugs reported for PG 7.2 cannot be reproduced at all in the older release (they affect newly added functionality) and, more importantly, 17 of the bugs do not cause failures

¹³ A `UNIQUE CONSTRAINT` means that within a set of columns no two values for different rows must be equal.

in the older release (2 of these bugs are Unreproduced in both releases). This means that the development of the newer release introduced many bugs in functionality that used to work correctly in the old release.

We also ran the old bugs in the new releases of the DBMS products to see how many were fixed. The results are given in the rightmost eight columns of Table 6 (full details are in [23]).

More PG 7.0 bugs were fixed in PG 7.2 than the number of bugs reported for IB that were fixed in FB. This may add an additional explanation to the results of the first half of Table 6, where we saw that some of the existing functionality of PG 7.0 had been “broken” by the attempted fixes in PG 7.2: there were more fixes integrated in this release, so the probability of breaking the existing functionality was also higher.

3.4.1 Implications for Fault Tolerance: The Open-Source Two-Version Combinations

Table 7 shows the results for all the bugs, from both studies, that could be run on the various open-source combinations.

The first two rows concern the pairs of different releases of the same DBMS product. For PostgreSQL we can see out of 93 bugs that caused failure in at least one of the releases 7.0 or 7.2 only 35 cause failures in both; 58 bugs cause failures in only one of the releases. So, using diverse releases of the same DBMS product in a fault-tolerant configuration, as discussed in Section 2, does provide some protection against upgrade problems and can help to assure higher dependability. However there are still many bugs causing failures in both releases of the same DBMS product:

- 57 in Interbase/Firebird
- 35 in PostgreSQL.

This can be compared with the four DBMS product pairs using different DBMS products (last four rows in Table 7), where we get at most 2 bugs that cause coincident failures. This is because:

- The IB 6.0 bug 223512(2) [23], which caused non-divergent coincident failure in IB 6.0 and PG 7.0, has been fixed in the newer releases of both DBMS products.
- The FB 1.0 bug 926001 [23], which causes coincident failure in the new releases FB 1.0 and PG 7.2, did not cause a failure in IB 6.0 and cannot be run in PG 7.0 (dialect-specific).

The main conclusion is to confirm the high level of fault diversity between these DBMS products, and thus potential advantages of a diverse redundant fault-tolerant server. Using different releases of the same DBMS product would also achieve dependability gains, but these still seem nowhere near as high as the gains that can be achieved by using diverse DBMS products.

4. DISCUSSION

The results presented in section 3 are intriguing and suggest that assembling a fault-tolerant database server from two or more of these OTS DBMS products could yield large dependability gains. But they are not definitive evidence. Apart from the sampling difficulties caused e.g. by lack of certain bug scripts, it is important to clarify to what extent

TABLE 6. THE RESULTS OF RUNNING THE NEW SCRIPTS REPORTED FOR FB AND PG 7.2 ON THE OLDER RELEASES (IB AND PG 7.0 RESPECTIVELY) AND THE BUGS REPORTED FOR THE OLD RELEASES ON THE NEW ONES
 ABBREVIATIONS: FB – FIREBIRD 1.0; IB – INTERBASE 6.0; PG 7.0 - PostgreSQL 7.0.0; PG 7.2 - PostgreSQL 7.2.

	FB	IB	PG 7.2	PG 7.0	PG 7.2	PG 7.0	FB	IB	IB	FB	PG 7.0	PG 7.2	PG 7.0	PG 7.2	IB	FB	
Total bug scripts	43	43	43	43	49	49	49	49	55	55	55	55	57	57	57	57	
Bug script cannot be run (Functionality Missing)	n/a	4	12	26	n/a	13	29	29	n/a	n/a	24	21	n/a	n/a	33	33	
Total bug scripts run	43	39	31	17	49	36	20	20	55	55	31	34	57	57	24	24	
Undecided performance	0	0	1	1	0	0	2	2	0	0	3	3	0	0	0	0	
No failure observed	4	6	29	16	4	17	17	17	8	33	27	31	5	40	24	24	
Failure observed	39	33	1	0	45	19	1	1	47	22	1	0	52	17	0	0	
Types of failures	Poor Performance	4	3	0	0	5	1	0	0	3	2	0	0	0	0	0	
	Engine Crash	6	6	0	0	10	3	0	0	7	2	0	0	11	2	0	
	Incorrect Result	Self-evident	7	6	0	0	13	8	1	1	4	2	0	0	14	6	0
		Non-self-evident	20	16	1	0	15	5	0	0	23	10	1	0	20	5	0
	Other	Self-evident	1	1	0	0	1	1	0	0	2	2	0	0	2	0	0
		Non-self-evident	1	1	0	0	1	1	0	0	8	4	0	0	5	4	0

TABLE 7. SUMMARY OF THE RESULTS OF BOTH STUDIES FOR OPEN-SOURCE TWO-VERSION COMBINATIONS
 (ABBREVIATIONS: S.E. – SELF-EVIDENT FAILURE; N.S.E. – NON-SELF-EVIDENT FAILURE)

Pairs of DBMS products	Total number of bug scripts run	Bugs scripts causing failure (in at least one DBMS product)	One out of two DBMS products failing		Both DBMS products failing						
			s.e.	n.s.e.	Non – Divergent				Divergent		
					All Demands		Some Demands		1 s.e. & 1 n.s.e.	2 s.e.	2 n.s.e.
					s.e.	n.s.e.	s.e.	n.s.e.			
FB 1.0 + IB 6.0	157	84	8	19	24	33	0	0	0	0	0
PG 7.2 + PG 7.0	164	93	33	25	20	15	0	0	0	0	0
FB 1.0 + PG 7.2	127	65	33	30	1	0	0	1	0	0	0
FB 1.0 + PG 7.0	106	65	34	30	1	0	0	0	0	0	0
IB 6.0 + PG 7.2	127	79	37	41	1	0	0	0	0	0	0
IB 6.0 + PG 7.0	106	77	39	37	0	1	0	0	0	0	0

our observations allow us to predict such gains. We gave a detailed discussion of the difficulties in [9, section 5]. In summary:

- the reports available concern *bugs*, not how many *failures* each caused. They do not tell us whether a bug has a large or a small effect on reliability, although the unknown faults – those that have not yet caused failures – would tend to have stochastically lower effect on reliability than those that did cause failures. A better analysis would be obtained from the actual failure reports (including failure counts), if available to the vendors. However, vendors are often wary of sharing such detailed dependability information with their customers.
- less than 100% of the failures that occur, and thus also of the bugs causing them, are reported. However, blatant failures are more likely to be reported than subtle (arguably more dangerous) failures. Therefore failure *underreporting* probably causes a bias towards *underestimating* the frequency of these subtler failures for which diversity would help.
- an organization needs to predict the dependability of its specific installation[s] of a diverse server, compared to a single DBMS product, which depends on the organization's (or each specific installation's) *usage profile*, which

differs – perhaps markedly – from the aggregate profile of the user population which generated the bug reports.

How can then individual user organizations decide whether diversity is a suitable option for them, with their specific requirements and usage profiles? As usual for dependability-enhancing measures, the cost is reasonably easy to assess: costs of the DBMS products, the required middleware, difficulties with client applications that require vendor-specific features, hardware costs, run-time cost of the synchronization and consistency enforcing mechanisms, and possibly more complex recovery after some failures. The gains in improved reliability and availability (fewer system failures and easier recovery from some failures, to be set against possible extra failures due to the added middleware), and possibly less frequent upgrades, are difficult to predict except empirically. In such cases using ballpark figures may provide useful guidelines: there are studies that suggest that the “Total Cost of Ownership” may exceed the initial investment by more than one order of magnitude, and the cost of recovery from failures is a major part of this [42]. This uncertainty will be compounded, for many user organizations, by the lack of trustworthy estimates of their baseline reliability with respect to subtle failures: databases are used with implicit

confidence that failures will be self-evident.

Despite all these uncertainties, for some users our evidence already means that a diverse server is a reasonable and relatively cheap precautionary choice, even without good predictions of its effects. These are users who have: serious concerns about dependability (e.g., high costs for interruptions of service or for undetected incorrect data being stored); applications which use mostly the core features common to multiple off-the-shelf DBMS products (recommended by practitioners to improve portability of the applications); modest throughput requirements for write statements, which make it easy to accept the synchronization delays of a fault-tolerant diverse server.

5. RELATED WORK

5.1 Fault Tolerance in Databases

Software fault tolerance has been thoroughly studied and successfully applied in many sectors, including databases. We already mentioned standard database mechanisms such as transaction “rollback and retry” and “checkpointing” which can be used to tolerate faults that are due to transient conditions. These techniques can be used with or without data replication in the databases. Data replication [13], [14], [10] solutions offered by OTS DBMS product vendors and their shortcomings were discussed at length in section 2.

5.2 Interoperability Between Databases

Due to the incompatibilities between the SQL “dialects” of different DBMS products we emphasized the need for SQL translators in the middleware of a diverse fault-tolerant server. Similar ideas have been applied for increasing the interoperability between the DBMS products [43], [44]: the grammar of a DBMS product is re-defined to make it compatible with that of another DBMS product, while keeping the core DBMS product engine unchanged.

Another solution which allows diverse replication (albeit with a minimal subset of SQL) is C-JDBC [45].

5.3 Empirical Studies of Faults and Failures

The usefulness of diversity depends on the frequency of those failures that cannot be tolerated without it. There have been comparatively few studies.

Gray studied the TANDEM NonStop system (with non-diverse replication) [20]. Over the (unspecified) measurement period, 131 out of 132 faults were “Heisenbugs” and thus tolerated. A later study of field software failures for the Tandem Guardian90 operating system [46] found that 82 % of the reported failures were tolerated. However, the others caused failure of both non-diverse processes in a Tandem process, and thus system failure.

Other related studies concern the determinism and fail-stop properties of database failures, but, like our study, they concern faults rather than failure measurements. A study [47] examined fault reports of three applications (Apache Web server, GNOME and MySQL DBMS product). Only a small fraction of the faults (5-14%) were Heisenbugs triggered by transient conditions, that would be tolerated by simple rollback retry. However, as the authors point out, the reason why they, like us, found few Heisenbugs, might

be that people are less likely to report faults that they cannot reproduce. Using fault injection the same authors also found [48] that a significant number of their injected faults (7%) violated the fail-stop model by writing incorrect data to stable storage. Although this fell to 2% when using the Postgres95 transaction mechanism, 2% is still high for applications with stringent reliability requirements.

5.4 Diversity with Off-The-Shelf Applications

Several research projects have addressed architectures supporting software fault tolerance for OTS software. Some have as their main aim intrusion tolerance, e.g.: HACQIT [49], which demonstrated diverse replication (with two OTS web servers - Microsoft’s IIS and Apache) to detect failures (especially maliciously caused ones) and initiate recovery; SITAR [50], an intrusion-tolerant architecture for distributed services and especially COTS servers; the Cactus architecture [3], intended to enhance survivability of applications which support diversity among application modules; DIT [51], an intrusion-tolerant architecture using diversity at several levels (hardware platform, operating system platform, and web servers); the MAFTIA [52] project delivered a reference architecture and supporting mechanisms. Others target fault tolerance against mainly accidental faults, e.g.: the BASE approach [53] focuses on supporting state recovery for diverse replicas of components via a common abstract specification of a common abstract state, the initial state value and the behavior of each component; the GUARDS [54] and Chameleon [55] architectures aim at supporting multiple application-transparent fault tolerance strategies using COTS hardware and software components.

In another example (Macromedia JRun) [56], uses diverse Java virtual machines for dealing with interoperability problems rather than for tolerating failures.

6. CONCLUSIONS

We have reported two studies with samples of bug reports for four popular off-the-shelf SQL DBMS products, plus later releases of two of them. We checked for bugs that would cause common-mode failures if the products were used in a diverse redundant architecture: such common bugs are rare. For most bugs, failures would be detected (and may be masked) by a simple two-diverse configuration using different DBMS products. In summary:

- out of the 273 bug scripts run in both our studies, we found very few bug scripts that affected two DBMS products and none that affected more than two.
- only five of these bug scripts caused identical, non-detectable failures in two DBMS products; of these five, one caused non-detectable failures on only a few among the demands affected.

The results of the second study, on later releases of the same products, substantially confirmed the general conclusions of the first study: one may conclude that the factors that make diversity useful do not disappear as the DBMS products evolve.

Using successive releases of the *same* product for fault tolerance also appeared useful, although less so. We found

a high level of fault diversity between successive releases of PostgreSQL: most of the old bugs had been fixed in the new release; many of the newly reported bugs did not cause failure (or could not be run at all) in the old release.

These results must be taken with caution, as discussed in section 4, and their immediate implications vary between users, but for some classes of DBMS product installations diversity could already be recommended as a prudent and cost-effective strategy. The topic of diversity with OTS software certainly deserves further study.

The need for middleware is an obstacle for users wishing to try out diversity in their applications (rudimentary solutions such as C-JDBC [45] only allow for the use of a minimal subset of SQL with diverse DBMS products). But our results provide a good business case for implementing the required middleware software.

The performance penalty due to controlling concurrency via the middleware would be a problem with write-intensive loads, but not if concurrent updates are rare [57].

Some other interesting observations include:

- there is strong evidence against the fail-stop failure assumption for DBMS products. The majority of bugs reported, for all products, led to "incorrect result" failures rather than crashes (64.5% vs 17.1% in our first study; 65.5% vs 19% in the second), despite crashes being more obvious to the user. Even though these are bug reports and not failure reports, this evidence goes against the common assumption that the majority of failures are engine crashes, and warrants more attention by users to fault-tolerant solutions, and by designers of fault-tolerant solutions to tolerating subtle and non fail-silent failures;
- it may be worthwhile for vendors to test their DBMS products using the known bug reports for other DBMS products. For example, in the first study we observed 4 MSSQL bugs that had not been reported in the MSSQL service packs (previous to our observation period). Oracle was the only DBMS product that never failed when running on it the reported bugs of the other DBMS products;

Future work that is desirable includes:

- statistical testing of the DBMS products to assess the actual reliability gains from diversity. We have run a few million queries with various loads, including ones based on the TPC-C benchmark, observing no failures (however, we found a significant potential for performance gain from using diverse servers [19], [57]). These results may not be particularly surprising, since these benchmarks use a limited set of well-exercised features of SQL servers. It would be interesting to repeat the tests with test loads that do not suffer from this limitation. However, these studies are likely to be most useful with reference to specific application environments for which the operational profile can be reasonably well approximated;
- developing the necessary middleware components for users to be able to try out data replication with diverse servers in their own installations. Lack of these components is the main practical obstacle in the way of the adoption and practical evaluation of these solutions. There are signs that some DBMS product vendors may also help with this problem: EnterpriseDB [43] and Fyralc [44] are Oracle-mode implementations based on Post-

greSQL and Firebird DBMS engines, respectively. With these solutions the problem with SQL dialects is significantly reduced.

ACKNOWLEDGMENT

This work was supported in part by the U.K. Engineering and Physical Sciences Research Council (EPSRC) via projects DOTS (Diversity with Off-The-Shelf components, grant GR/N23912/01) and DIRC (Interdisciplinary Research Collaboration in Dependability, grant GR/N13999/01) and by the European Union Framework Program 6 via the ReSIST Network of Excellence (Resilience for Survivability in Information Society Technologies, contract IST-4-026764-NOE). We thank Bev Littlewood, Peter Bishop and David Wright for their comments on earlier versions of this paper.

REFERENCES

- [1] P. Popov, L. Strigini and A. Romanovsky, "Diversity for Off-The-Shelf Components", *Proc. IEEE DSN'00 - Fast Abstracts supplement*, New York, NY, USA, pp. B60-B61, 2000.
- [2] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou and T.E. Uribe, "An Adaptive Intrusion-Tolerant Server Architecture", http://www.sdl.sri.com/users/valdes/DIT_arch.pdf, 1999.
- [3] M.A. Hiltunen, R.D. Schlichting, C.A. Ugarte and G.T. Wong, "Survivability Through Customization and Adaptability: The Cactus Approach", *Proc. DARPA Information Survivability Conference & Exposition*, 2000.
- [4] L. Strigini, "Fault Tolerance Against Design Faults," in *Dependable Computing Systems: Paradigms, Performance Issues, and Applications*, H. Diab and A. Zomaya, Eds.: J. Wiley & Sons, pp. 213-241, 2005.
- [5] P. Popov, L. Strigini, S. Riddle and A. Romanovsky, "Protective Wrapping of OIS Components", *Proc. 4th ICSE Workshop on Component-Based Software Engineering: Component Certification and System Prediction*, Toronto, Canada, 2001.
- [6] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil and P. O'Neil, "A Critique of ANSI SQL Isolation Levels", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, San Jose, CA, USA, 1995.
- [7] A. Fekete, D. Liarokapis, E. O'Neil, P. O'Neil and D. Shasha, "Making Snapshots Isolation Serializable", <http://www.cs.umb.edu/~isotest/snaptest/snaptest.pdf>, 2000.
- [8] F. Schneider, "Byzantine Generals in Action: Implementing Fail-Stop Processors", *ACM TOCS*, 2(2), pp. 145-154, 1984.
- [9] I. Gashi, P. Popov and L. Strigini, "Fault Diversity Among Off-The-Shelf SQL Database Servers", *Proc. IEEE DSN'04*, Florence, Italy, pp. 389-398, 2004.
- [10] P.A. Bernstein, V. Hadzilacos and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Reading, Mass., Addison-Wesley, 1987.
- [11] M. Weismann, F. Pedone and A. Schiper, "Database Replication Techniques: a Three Parameter Classification", *Proc. IEEE SRDS'00*, Nurnberg, Germany, pp. 206-217, 2000.
- [12] F. Pedone and S. Frolund, "Pronto: A Fast Failover Protocol for Off-the-shelf Commercial Databases", *Proc. IEEE SRDS'00*, Nurnberg, Germany, pp. 176-185, 2000.
- [13] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme and G. Alonso, "MIDDLE-R: Consistent Database Replication at the Middleware Level", *ACM TOCS*, 23(4), pp. 375-423, 2005.
- [14] Y. Lin, B. Kemme, M. Patiño-Martínez and R. Jiménez-Peris, "Middleware based Data Replication providing Snapshot Isolation", *Proc. ACM SIG-*

- MOD Int. Conf. on Management of Data*, Baltimore, MD, USA, pp. 419-430, 2005.
- [15] R. Jimenez-Peris and M. Patino-Martinez, "D5: Transaction Support", ADAPT Middleware Technologies for Adaptive and Composable Distributed Components Deliverable IST-2001-37126, 21 March, 2003.
- [16] R. Jimenez-Peris, M. Patino-Martinez, G. Alonso and B. Kemme, "Scalable Database Replication Middleware", *Proc. 22nd IEEE Int. Conf. on Distributed Computing Systems*, Vienna, Austria, pp. 477-484, 2002.
- [17] H. Sutter, "SQL/Replication Scope and Requirements Document", *ISO/IEC JTC 1/SC 32 Data Management and Interchange WG3 Database Languages*, H2-2000-568, 2000.
- [18] B. Kemme and G. Alonso, "Don't be Lazy, be Consistent: Postgres-R, a New Way to Implement Database Replication", *Proc. Int. Conf. on Very Large Databases (VLDB)*, Cairo, Egypt, 2000.
- [19] I. Gashi, P. Popov, V. Stankovic and L. Strigini, "On Designing Dependable Services with Diverse Off-The-Shelf SQL Servers", in *Architecting Dependable Systems II*, vol. 3069, LNCS, R. de Lemos, Gacek, C., Romanovsky, A., Ed.: Springer-Verlag, pp. 191-214, 2004.
- [20] J. Gray, "Why Do Computers Stop and What Can be Done About it?" *Proc. 6th Int. Conf. on Reliability and Distributed Databases*, 1987.
- [21] P.E. Ammann and J.C. Knight, "Data Diversity: An Approach to Software Fault Tolerance", *IEEE TC*, C-37(4), pp. 418-425, 1988.
- [22] I. Gashi and P. Popov, "Rephrasing Rules for Off-The-Shelf SQL Database Servers", *Proc. IEEE EDCC-6*, Coimbra, Portugal, pp. 139-148, 2006.
- [23] I. Gashi, "Fault Diversity Among Off-The-Shelf SQL Database Servers: Complete Results From Two Studies", <http://www.csr.city.ac.uk/people/ilir.gashi/DBMSBugReports/>, 2006
- [24] J.E. Cook and J.A. Dage, "Highly Reliable Upgrading of Components", *Proc. Int. Conf. on Software Engineering (ICSE '99)*, L.A., CA, USA pp. 203-212, 1999.
- [25] A.T. Tai, K.S. Tso, L. Alkalai, S.N. Chau and W.H. Sanders, "Low-Cost Error Containment and Recovery for Onboard Guarded Software Upgrading and Beyond", *IEEE TC*, C-51(2), pp. 121-137, 2002.
- [26] D. Bakken, "Middleware: What it is, and How it Enables Adaptivity and Dependability", *Proc. 43 Meeting of IFIP WG 10.4 Dependable Computing and Fault Tolerance*, Santa Maria, Sal Island, Cape Verde, pp. 13-40, 2003.
- [27] T. Anderson and P.A. Lee, "Fault Tolerance: Principles and Practice (Dependable Computing and Fault Tolerant Systems, Vol 3)", Springer Verlag, 1990.
- [28] P. Popov, L. Strigini, A. Kostov, V. Mollov and D. Selensky, "Software Fault-Tolerance with Off-the-Shelf SQL Servers", *Proc. 3rd Int. Conf. on COTS-based Software Systems (ICBSS'04)*, Redondo Beach, CA, USA, pp. 117-126, 2004.
- [29] R. Jimenez-Peris, M. Patino-Martinez and G. Alonso, "An Algorithm for Non-Intrusive, Parallel Recovery of Replicated Data and its Correctness", *Proc. IEEE SRDS'02*, Osaka, Japan, pp. 150-159, 2002.
- [30] F. Di Giandomenico and L. Strigini, "Adjudicators for Diverse-Redundant Components", *Proc. IEEE SRDS'90*, Huntsville, AL, USA, pp. 114-123, 1990.
- [31] D.M. Blough and G.F. Sullivan, "Voting Using Predispositions", *IEEE TR*, R-43(4), pp. 604-616, 1994.
- [32] B. Parhami, "Voting: A Paradigm for Adjudication and Data Fusion in Dependable Systems", in *Dependable Computing Systems: Paradigms, Performance Issues, and Applications*, H.B. Diab and A. Y. Zomaya, Eds, 2005.
- [33] Y. Bao, X. Sun and K.S. Trivedi, "A Workload-based Analysis of Software Aging and Rejuvenation", *IEEE TR*, R-54(3), pp. 541-548, 2005.
- [34] J. Gray and A. Reuter, "Transaction Processing : Concepts and Techniques", Morgan Kaufmann, 1993.
- [35] K.S. Tso and A. Avizienis, "Community Error Recovery in N-Version Software: A Design Study with Experimentation", *Proc. IEEE FTCS-17*, Pittsburgh, PA, USA, pp. 127-133, 1987.
- [36] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance", *Proc. Third Symp. on Operating Systems Design and Implementation*, New Orleans, LA, USA, pp. 173-186, 1999.
- [37] P. Frankl, D. Hamlet, B. Littlewood and L. Strigini, "Evaluating Testing Methods by Delivered Reliability", *IEEE TSE*, SE-24(8), pp. 586-601, 1998.
- [38] SourceForge, "Interbase (Firebird) Bug tracker", http://sourceforge.net/tracker/?atid=109028&group_id=9028&func=browse, 2006.
- [39] PostgreSQL, "PostgreSQL Bugs Mailing List Archives", <http://archives.postgresql.org/pgsql-bugs/>, 2006.
- [40] Microsoft, "List of Bugs Fixed by SQL Server 7.0 Service Packs", <http://support.microsoft.com/default.aspx?scid=kb;EN=US;313980>, 2006
- [41] Oracle, "Oracle Metalink", http://metalink.oracle.com/metalink/plsql/ml2_gui.startup, 2006.
- [42] D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kycyman, M. Merzbacher, D. Oppenheimer, N. Sastri, W. Tetzlaff, J. Traupman and N. Treuhaf, "Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques and Case Studies", UC Berkeley Computer Science: 16 2002.
- [43] EnterpriseDB, "EnterpriseDB", <http://www.enterprisedb.com/>, 2006.
- [44] Janus-Software, "Fyracle", http://www.janus-software.com/fb_fyracle.html, 2006.
- [45] ObjectWeb, "C-JDBC", <http://c-jdbc.objectweb.org/>, 2006
- [46] I. Lee and R.K. Iyer, "Software Dependability in the Tandem GUARDIAN System", *IEEE TSE*, 21(5), pp. 455-467, 1995.
- [47] S. Chandra and P.M. Chen, "Whither Generic Recovery from Application Faults? A Fault Study using Open-Source Software", *Proc. IEEE DSN'00*, NY, USA, pp. 97-106, 2000.
- [48] S. Chandra and P.M. Chen, "How Fail-Stop are Programs", *Proc. IEEE FTCS-28*, Munich, Germany, pp. 240-249, 1998.
- [49] J. Reynolds, J. Just, E. Lawson, L. Clough, R. Maglich and K. Levitt, "The Design and Implementation of an Intrusion Tolerant System", *Proc. IEEE DSN'02*, Washington, D.C., USA, pp. 285-292, 2002.
- [50] F. Wang, F. Gong, C. Sargor, K. Goseva-Popstojanova, K. Trivedi and F. Jou, "SITAR: A Scalable Intrusion-Tolerant Architecture for Distributed Services", *Proc. 2001 IEEE Workshop on Information Assurance and Security*, West Point, NY, USA, 2001.
- [51] A. Valdes, M. Almgren, S. Cheung, Y. Deswarte, B. Dutertre, J. Levy, H. Saidi, V. Stavridou and T.E. Uribe, "An Architecture for an Adaptive Intrusion-Tolerant Server", in *LNCS 2845 - Selected Papers from 10th International Workshop on Security Protocols '02*, B. Christianson, Crispo, B., Malcolm, J. A., Roe, M., Ed.: Springer, pp. 158-178, 2003.
- [52] M. Dacier (Editor), "Design of an Intrusion-Tolerant Intrusion Detection System", MAFTIA deliverable D10, <http://www.maftia.org/deliverables/D10.pdf>, 2002
- [53] M. Castro, R. Rodrigues and B. Liskov, "BASE: Using Abstraction to Improve Fault Tolerance", *ACM TOCS*, 21(3), pp. 236-269, 2003.
- [54] D. Powell, J. Arlat, L. Beus-Dukic, A. Bondavalli, P. Coppola, A. Fantechi, E. Jenn, C. Rabejac and A. Wellings, "GUARDS: A Generic Upgradable Architecture for Real-Time Dependable Systems", *IEEE TPDS*, 10(6), pp. 580-599, 1999.
- [55] Z.T. Kalbarczyk, R.K. Iyer, S. Bagchi and K. Whisnant, "Chameleon: A Software Infrastructure for Adaptive Fault Tolerance", *IEEE TPDS*, 10(6), pp. 560-579, 1999.
- [56] Adobe, "Macromedia JRun", <http://www.adobe.com/products/jrun/productinfo/overview/>, 2006
- [57] V. Stankovic and P. Popov, "Improving DBMS Performance through Diverse Redundancy", *Proc. IEEE SRDS'06*, Leeds, UK, pp. 391-400, 2006.

Modeling of Reliable Messaging in Service Oriented Architectures³

László Gönczy¹ and Dániel Varró²

*Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary*

Abstract

Due to the increasing need of highly dependable services in Service-Oriented Architectures (SOA), service-level agreements include more and more frequently such traditional aspects as security, safety, availability, reliability, etc. Whenever a service can no longer be provided with the required QoS, the service requester need to switch dynamically to a new service having adequate service parameters. In the current paper, we propose a metamodel to capture such parameters required for reliable messaging in services in a semi-formal way (as an extension to [1]). Furthermore, we incorporate fault-tolerant algorithms into appropriate reconfiguration mechanisms for modeling reliable message delivery by graph transformation rules.

Key words: Service Oriented Architecture, Graph Transformation, Reliable Messaging,

1 Introduction

Service-Oriented Architectures (SOA) provide a flexible and dynamic platform for implementing business services. The main business-level driver of the SOA paradigm is *componentization*, which raises the level of abstraction from objects to services in the design process of distributed applications. The main architectural-level driver of the SOA paradigm is to provide a common middleware framework for dynamic discovery, interaction and reconfiguration of service components independently of the actual business environment.

Due to the increasing need of highly dependable services, service-level agreements include more and more frequently such traditional (non-functional) aspects as security, safety, availability, reliability, etc. The general idea is that

¹ Email: gonczy@mit.bme.hu

² Email: varro@mit.bme.hu

³ This work was partially supported by the SENSORIA European project (IST-3-016004).

whenever a service can no longer be provided with the required QoS, the service requester needs to switch dynamically to a new service having adequate service parameters.

In an ideal scenario of using service-level agreements, designers only specify the requirements for a specific service, and reconfigurations aiming to maintain the required QoS parameters are handled automatically by the underlying service middleware platform. Therefore, the service requestor does not need to be adapted explicitly (i.e. on the code level) to the evolution of the environment.

Recently, the identification of non-functional parameters of services have been addressed by various XML-based standards related to web services (such as WS-Reliable Messaging, WS-Reliable Messaging Policies, etc.). A focal topic in many of these standards is related to reliable messaging between services, where the delivery of a message can be guaranteed by the underlying platform by appropriate reconfiguration mechanisms. In contrast to the *specification* of these reliability service properties, currently only very experimental solutions exist in the industry (such as RAMP-Toolkit [13] by IBM or RM4GS [14] by a consortium led by Fujitsu-Siemens, Hitachi and NEC) that actually implement these reconfigurations in order to maintain the required level of reliability.

In the paper, we facilitate the use of a precise model-based approach for the development of high-level, reconfiguration mechanisms required for reliable messaging in the underlying service middleware. Our long term goal is automatically derive implementations of reliable messaging on various existing platforms based directly upon provenly correct dynamic reconfiguration mechanisms.

In the current paper, we conceptually follow [1] where a semi-formal platform-independent and a SOA-specific metamodel (ontology) was developed to capture service architectures on various levels of abstraction in a model-driven service development process. Furthermore, reconfigurations for service publishing, querying and binding were captured by graph transformation rules [4], which provide a formal, rule and pattern-based specification formalism widely used in various application areas. This combination of metamodeling and graph transformation rules fits well to a model-based development process for service middleware.

This paper extends the core metamodel defined in [1] (and overviewed in Sec.3) by a new package for reliable messaging (Sec. 4.2). Moreover, we provide new reconfiguration primitives for reliable message delivery in the form of graph transformation rules (Sec. 5.2) by integrating dependability techniques [10].

2 Service Reconfigurations for Reliable Messaging: An Overview

Our overall research objectives towards provenly correct service reconfigurations for reliable messaging follows the SENSORIA approach [16], and it is sketched in Fig. 1.

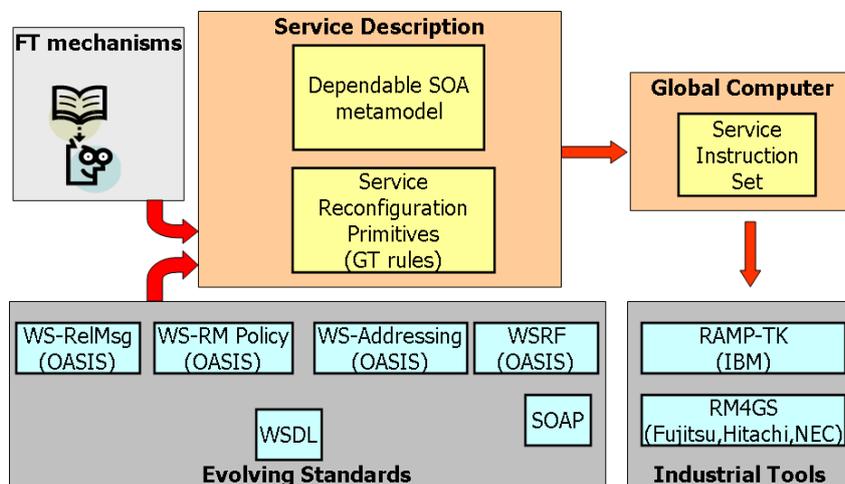


Fig. 1. Towards Provenly Correct Service Reconfigurations for Reliable Messaging

- A *dependability extension of the core SOA metamodel* will be developed which is synthesized from existing standards related to web services and other application areas. The current paper provides a first step towards this by focusing on service parameters for reliable messaging.
- *High-level reconfiguration primitives* will be defined to capture dependable services in the form of graph transformation rules by integrating traditional fault-tolerance techniques into SOA. In the current paper, basic reconfiguration steps are identified for providing reliable messaging between services.
- *Formal analysis* will be carried out in order to justify the correctness of reconfigurations.
- A new kind of virtual machine (called a *global computer*) is envisaged with a special *instruction set* tailored to dependable service reconfigurations.
- Finally, this instruction set will be *mapped to existing technologies* dedicated to the development of reliable web services (such as RAMP [13] or RM4GS [14])

3 Core SOA Metamodel

To illustrate the problem domain with a small but practical example, the case study of a *course management system* is introduced.

Let us consider a university where the students can perform their administrative tasks (for instance, signing for courses and exams) online. Hereby

we model the *exam management service* within this course management system. The exam management component offers services to students, teachers and other employees of the university like the administrative staff. Students can sign up for or cancel an exam while teachers can submit the result of the exams. Administrative staff can perform queries against the exam database (for instance, to retrieve the course with the highest failure ratio).

The main architectural concepts of the domain of service-oriented architectures are captured by a corresponding metamodel. The metamodel of "core" SOA functionality is shown in Fig. 2. It is based on the metamodel presented in [1], with a minor modification of merging both the structural and dynamic aspects into a single package.

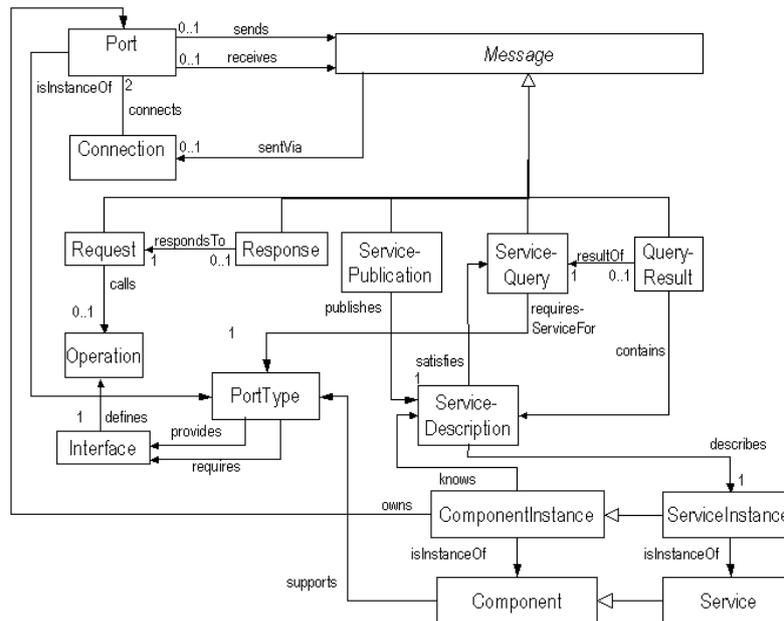


Fig. 2. Core metamodel of SOA

The core model to service-oriented architectures consists of the following elements:

- A *component* is a basic "module" in the system which provides a service. In the online course management system, for instance, **ExamAdministration**, **ExamRegistration**, **ExamQuery** will be such components.
- A *service* is a set of well-defined functionality. In our case, there will be three services: **ExamAdministrationService** for educational staff to create new exams, set the parameters of exams (such as date, limit, etc.) and to upload the results of the exams; **ExamRegistrationService** for students to register themselves to exams (and to cancel registrations, if necessary) and **ExamQuery** for the administrative staff to retrieve statistics on exams. These will be provided by the three components, respectively.
- A *port* is the communication "endpoint" of the service, with a set of abstract

operations and messages. For instance, `ExamRegistrationPort` is the interface of `ExamRegistrationService`.

- A *connection* denotes a bidirectional channel between two ports at run-time.
- An *operation* is an "atomic" action with input and output messages. There can be multiple operations defined on the same port. `SignUp` is an operation, defined on `ExamRegistrationPort`.
- A *message* is a set of parameters with pre-defined types. For instance, `ExamDescription` message may have parameters like `studentName`, `examId`, etc. The abstract message class is refined into the following subtypes of messages: *response*, *request*, *service publication*, *query and query result*.
- A *service description* is a descriptor file containing all necessary information about the runtime cooperation with the service, such as description of port, operations, messages, etc.

4 Extensions for Reliable Messaging in Web Services

4.1 Non-functional Requirements in Existing Web Service Technologies

Although there exist some initiatives to define the so-called "non-functional" properties of services, such as Web Services Modeling Ontology [18], W3C Web Services Architecture [17], DublinCore Metadata for ServiceDiscovery [3], the terminology is still ambiguous.

To illustrate the modeling of non-functional properties by a practical and simple example, hereby we present a model-based reconfiguration for reliable messaging to tolerate communication faults. As the consumers of the Web services are not aware of the details of underlying network protocol, the semantics of the message delivery has to be specified at the application level as requirements for reliable messaging. This needs a platform-independent representation of message attributes, which is reflected by a number of emerging standards, such as [20] and [19], which are converging to each other [21]. Some reference implementations for popular application servers like IBM WebSphere or Apache Tomcat are available.

These industrial standards and initiatives usually suppose that the service provide signs a contract with each client about the Quality of Service, measured in terms such as average response time, minimal throughput, type of message delivery, etc. These contracts are typically identical for classes of similar clients (roles), for instance, Golden User, Business Partner, Individual Customer, etc. The runtime service instances send their messages according to these contracts, and the additional information, regarding these non-functional aspects, is hidden from the application layer, so that the modification of the original clients on the consumers' side is not necessary. The additional information is handled by components aware of reliability attributes, called "*Reliable Message Endpoints*". Technologically speaking, the header of SOAP

envelopes is extended with some attributes by a "Reliable Message Endpoint" on the provider's side, which are then removed from the messages by another "Reliable Message Endpoint" at the client side. Since the concrete format of these attributes in message headers is out of our scope, here we model an abstract description, which, however, will be mapped to existing technologies using model transformation techniques in the future.

4.2 Metamodel extensions for reliable messaging in services

Now we extend the core SOA metamodel of [1] to capture properties of reliable messaging between services. After enriching the domain metamodel, our long term goal is to define a corresponding UML profile to provide extensions to the UML language tailored to a specific application domain by introducing domain concepts, attributes and relations in the form of stereotypes and tagged values. However, the current paper only focuses on designing extensions for reliable messaging in the SOA metamodel.

In the current paper, we first derive a subclass from SOA element in the reliable SOA metamodel, and then create an association from the child class (e.g. `RelMsgEnvelope`) to the parent class (e.g. `Message`) in addition. As a result, original SOA reconfiguration rules defined in [1] are still applicable with the extensions for the reliable messaging metamodel, thus we can handle messaging between services on the communication level in the same way as before. Furthermore, the original messages are kept but wrapped into an envelope by introducing a new association.

The extensions of the SOA metamodel for reliable messaging is presented in Fig. 3:

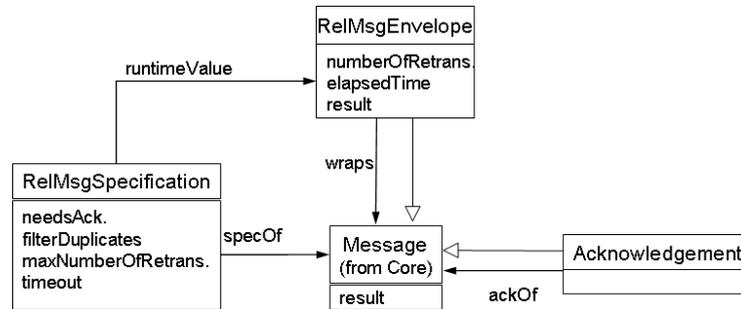


Fig. 3. Metamodel of Reliability Extensions

- `RelMsgSpecification` is a class for specifying the requirements for reliable messaging between SOA services (see association `specOf`).
 - Attribute `needsAck` is a boolean value to express if an acknowledgement should be sent to a message. If an acknowledgement arrives to the sender for a message, then it is guaranteed that the message is sent at least once.
 - Attribute `filterDuplicates` is a boolean value to express that a message should be accepted and processed by the receiver at most once.

- Attribute **timeout** is a timer constraint which specifies how much does the sender waits for the acknowledgement of a message before retransmission.
- Attribute **maxNumberOfRetrans** is an integer which puts an upper limit how many times a message can be retransmitted by the sender due to the lack of acknowledgement from the receiver.
- **RelMsgEnvelope** is a subclass of core SOA **Message** which serves as an envelope for wrapping up the real message to be sent (**wraps**).
 - Attribute **numberOfRetrans** is a serial number for the envelope which is increased by one each time the same message is retransmitted.
 - Attribute **timeElapsed** denotes the time elapsed since the (last) transmission of a message.
- **Acknowledgement** is a subclass of core SOA **Message** which denotes an acknowledgement sent in response to a message.

As this extension reflects to existing standards, implementations of models can be derived following the Model Driven Architecture (MDA, [12]) approach. Runtime values of attributes will be generated from XML descriptors. The automated generation of XML descriptors from high level models is part of our future research.

In the course management application, let us consider the scenario of signing up for an exam. The two communicating components are **ClientApp** and **ExamRegistration**. The client signs up for a particular exam by sending a message to the instance port of **ExamRegistrationProviderPT** port type. This operation needs an *ExactlyOnce* messaging semantics, denoted by the attributes of the reliability specification of the message. Fig. 4 shows the relevant part of the instance graph of the system.

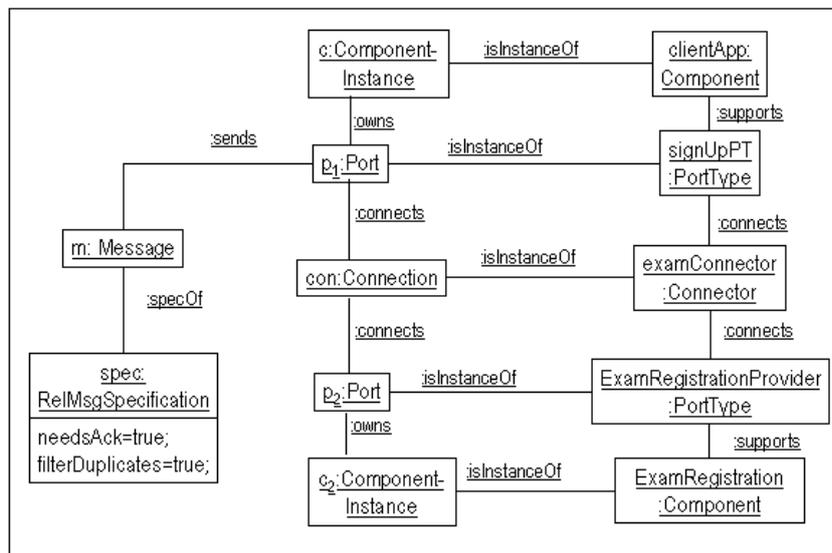


Fig. 4. Instance graph of the exam management system

Message sending operation will be executed by the application of transform-

mations on the instance graph as described in Sect. 5.

5 Reconfiguration for Reliable SOA Messaging by Graph Transformation

We now propose to describe the reconfiguration mechanisms of reliable SOA messaging by graph transformation rules (conceptually following the approach presented in [1]).

5.1 Overview of graph transformation

A main benefit of using graph transformations as a formal specification paradigm for capturing reconfiguration rules is that they are visual, intuitive, therefore they can be understood by service engineers as well. The interested reader may find a detailed theoretical discussion of graph transformation in [4], here we present just a brief overview on it.

Furthermore, graph transformation allows dynamic metamodeling [7] in a certain domain. The high-level (ontological) concepts are visualized as UML class diagrams while graph patterns are considered to be UML object diagrams to express that concrete models are instances (objects) of the meta-model (classes). This combines the advantage of precise modeling and visual design, as the ontology "behind" the class diagram defines the semantics of the model, while the graph transformation rules, interpreted on concrete model instances, can be designed visually by existing graph transformation tools.

A graph transformation rule consists of a *Left Hand Side (LHS)*, a *Right Hand Side (RHS)* and optionally a *Negative Application Condition (NAC)*. The *LHS* is a graph pattern consisting of the *mandatory* elements which prescribes a precondition for the application of the rule. The *RHS* is a graph pattern containing all elements which should be present after the application of the rule. Elements in the $RHS \cap LHS$ are unchanged after the execution of the transformation, elements in $LHS \setminus RHS$ are deleted while elements in $RHS \setminus LHS$ are newly created by the rule. The fulfillment of the negative condition prevents the rule from being executed on the particular matching. Furthermore, graph transformation rules frequently allow the use of *attribute conditions (constraints)* which restrict attributes of the matched nodes, and *attribute assignments*, which may describe the updates of certain attributes as a result of rule application. Hereby we follow the Double Pushout (DPO) approach [4] for the semantics of graph transformation.

A *Graph Transformation System, GTS* consists of a graph instance and the transformation rules. The execution of a GTS is nondeterministic, since the next rule to be applied and the matching on which a rule is applied is not restricted (by default) by additional control information.

In this paper, we use a compact visualization of graph transformation rules (first introduced in the Fujaba framework [6]), when the entire rule is merged

into a single pattern. Newly created elements are denoted by $\{new\}$ and by slashed lines while deleted elements have a grey background and a $\{deleted\}$ tag. Elements in the subset of the *LHS* and the *RHS* are visualized normally, and elements of NAC are crossed out. In the current paper, the $\{new\}$ tag implicitly implicates a negative condition as well, which prevents the rule from creating infinite number of new elements on the same matching (in the case of messaging, the same message is received only once).

Rules of Fig. 5 (which is a simplified version of rules presented in [1]) illustrate message sending and receiving in Service Oriented Architectures with the "traditional" and the compact visualization style.

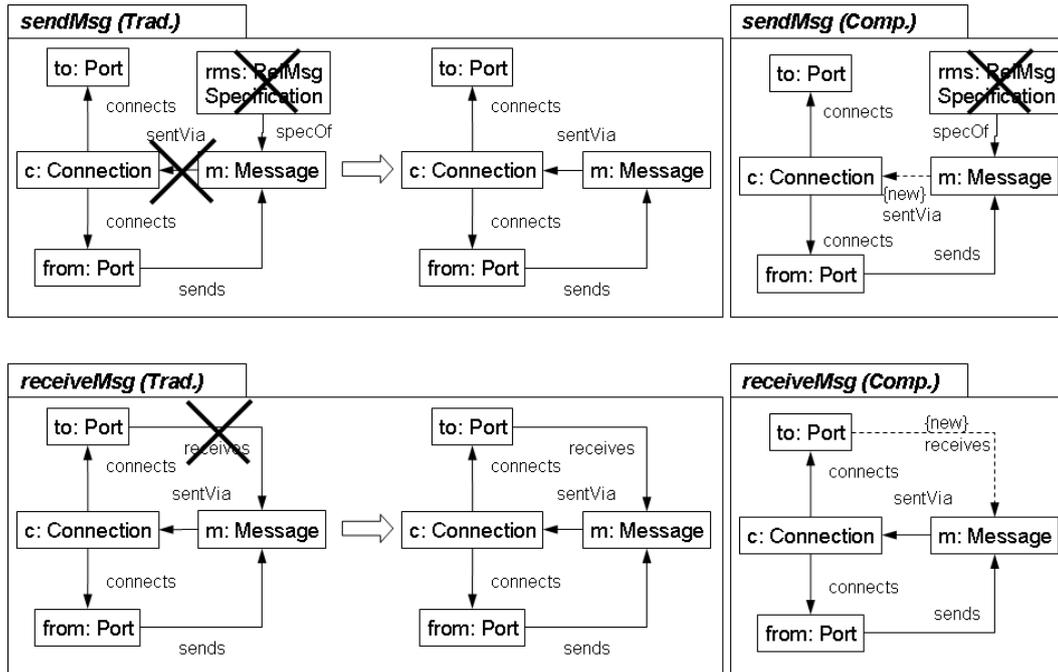


Fig. 5. Sending and receiving a message

The rules in Fig. 5 capture the operations of the basic messaging. Two services can communicate if there is an open connection between them. The message is first sent via the connection (a new `sentVia` association is created). The service provider can receive the message only once, this is implicated by the negative application condition of the receive rule.

5.2 Reconfiguration Rules

The reliable messaging can be assured by the following reconfiguration rules captured by graph transformation.

First, the normal messages have to be packed into and wrapped from envelopes (as in the case of present reliable messaging technologies). Thus, the messages are wrapped up in the sender side instead of being transmitted (Fig. 6) and envelopes are opened before receiving their content at the receiver

side Fig. 7. As the type of the message is of specified, these rules will match for instances of every subclass of message class with a reliability specification. Thus, reliable messaging is also provided for asynchronous service invocations, discovery queries, etc.

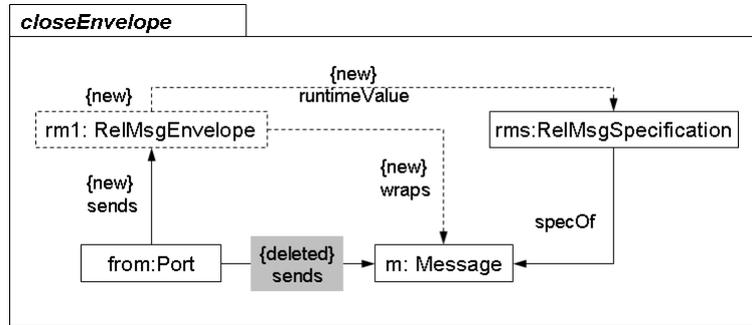


Fig. 6. Wrapping a Message into an Envelope

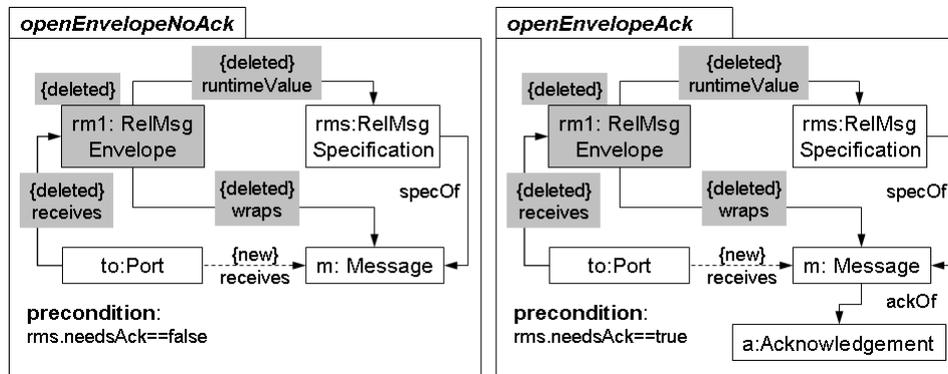


Fig. 7. Opening an Envelope

At the sender side, there are basically two message sending modes, depending on the value of the `needsAck` parameter of the `ReliableMessageSpecification` object describing the requirements for messaging.

At least once message delivery

If this parameter is `true`, reliable message sending required for a particular message, which corresponds to the *AtLeastOnce* messaging semantics. In this case, the sender will wait for an acknowledgement and consider the transmission of a message successful only if the acknowledgement arrives within the timeout interval. The rule of the successful message transmission (more precisely, the arrival of an acknowledgement in time) is shown in Fig. 8

On the other hand, if the acknowledgement does not arrive in time, then the next action (i.e. the next rule to be applied) depends on the number of retransmitted messages. If the retransmission number of a particular message is smaller than the allowed (precondition of rule *RetransmitMsg*), then a new instance of the `ReliableMsgEnvelope` class is created and sent with the same

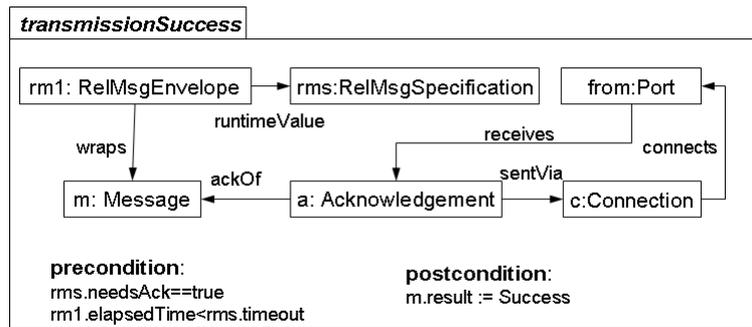


Fig. 8. Acknowledgement arrives in time

content and a higher retransmission number (rule *RetransmitMsg*). If the same message content cannot be sent again (precondition of rule *TransmissionFailure*), then the transmission of the message is considered to be failed. Note that if no acknowledgement is needed, then no additional rules are applied at message sending, only the core "SendMsg" rule matches the instance graph.

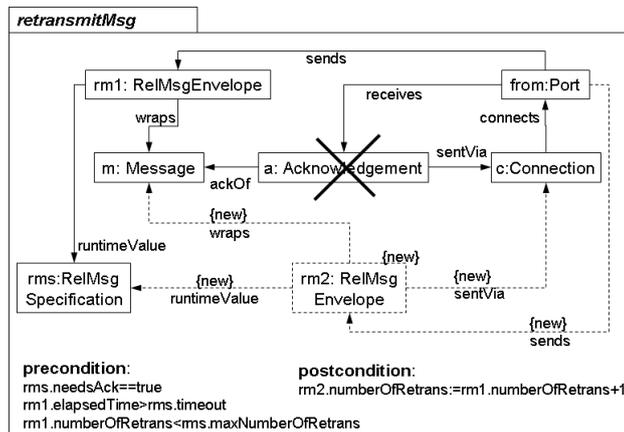


Fig. 9. Retransmission of a Message if Timeout Exceeded

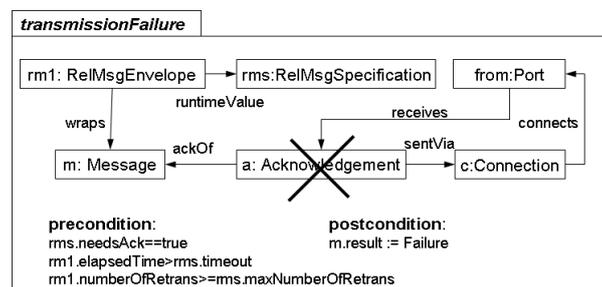


Fig. 10. Failure of Message Transmission

On the receiver side, the messages are acknowledged if needed (see Fig. 11), otherwise the core *ReceiveMsg* rule is applied.

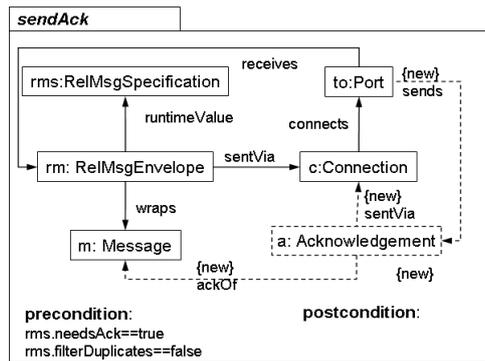
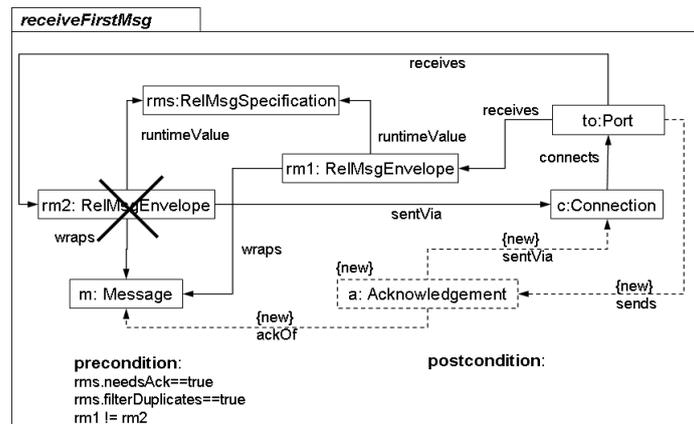


Fig. 11. Sending an Acknowledgement to an incoming message

Exactly once message delivery

Some applications may need the guarantee of the *ExactlyOnce* semantics, which prescribes acknowledgements and filtering (`needsAck` and `filterDuplicates` are set to true, respectively).

At this communication mode, if the first instance arrives (no previous message has been received with the same content), then the message is received and an acknowledgement is created and sent back, as shown in Fig. 12.

Fig. 12. Receiving the first message instance with the *ExactlyOnce* semantics

If the acknowledgement of a message is lost (due to some network error), then the sender will retransmit the same message after the timeout exceeds. In this case, a duplicate message will be received at the `from` Port. If the duplicates are filtered, then the arriving duplicate message is dropped but an acknowledgment is still created, as shown in Fig. 13.

In the case of registration for an exam, the successful registration may correspond to the following rule application sequence: *closeEnvelope*, *sendMsg*, *sendAck*, *openEnvelope*, *receiveMsg*, and *transmissionSuccess*.

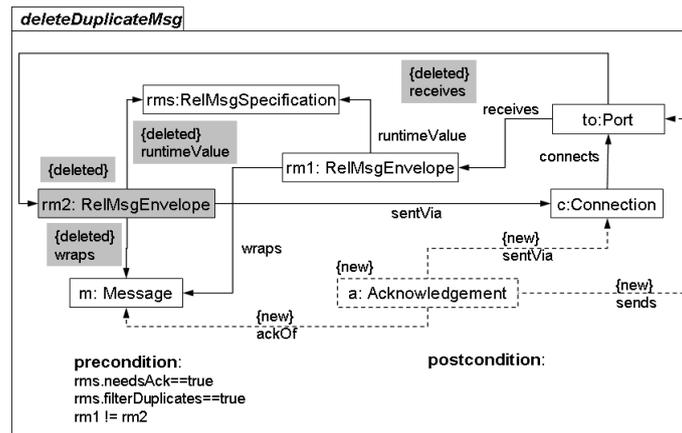


Fig. 13. Deleting duplicates

6 Related work

Related works in this field usually concentrate either on describing the non-functional attributes of services, or on visual modeling of dynamic aspects of Service Oriented Architectures. Our work is based on the approach of [1]. [2] describes the application of graph transformations in the runtime matching of behavioral Web service specifications. In [8], the conformance testing of Web services is based on graph transformations, focusing on the automated test case generation. However, none of these works discusses the aspects of reliable messaging. Our aim was to utilize the benefits of this approach by extending the metamodel and the transformation rules.

Graph transformation is used as a specification technique for dynamic architectural reconfigurations in [5] using the algebraic framework CommUnity. Hirsch uses graph transformations over hypergraphs in [9] to specify run-time interactions among components, reconfigurations, and mobility in a given architectural style. However, the problem of reliable messaging in SOA is not addressed in either case.

A profile for reliability was designed for J2EE applications in [15]. Our work is different in the sense that we introduce a formal operational semantics of reliability messaging mechanisms which can provide basis for the underlying SOA middleware in an application independent way.

In [11], a pattern based specification and run-time validation approach is presented for interaction properties of web services using a semantic web rule language (SWRL). These patterns include constraints (requirements) on service invocations including *at-most-n* and *at-least-n* message delivery. This approach only reports run-time violation of the constraints, while our overall goal is to guarantee the delivery of messages with appropriate reliability semantics by the underlying middleware.

In the industrial field, there are existing and emerging specifications and technologies like [19], [20], [21]. However, their use is still ad-hoc and no model-based design-time support is available for reliable messaging.

7 Conclusion

In this paper we proposed an extension to the core SOA metamodel of [1] and a technique to capture the reconfiguration mechanisms to enhance the development of more robust SOA middleware. The main advantage of our solution is its seamless integration with the previous initiative: core SOA reconfiguration mechanisms of [1] are directly applicable without changes, furthermore, the original messages are kept unaltered by the proposed wrap up mechanism.

We are currently working on the formal verification of the correctness of the proposed reconfiguration mechanisms using existing verification tools for graph transformation systems. As the next step in the future, we plan to implement the automatic generation of runtime implementation in existing middleware and to create test cases for reliable messaging.

References

- [1] Baresi, L., R. Heckel, S. Thöne and D. Varró, *Style-Based Modeling and Refinement of Service-Oriented Architectures*. To appear in Journal of Software and Systems Modelling, 2006.
- [2] A. Cherchago and R. Heckel. *Specification Matching of Web Services Using Conditional Graph Transformation Rules*. In Proc. of International Conference on Graph Transformations, 2004, LNCS Vol. **3256**, Springer, pp. 304-318.
- [3] DublinCore Metadata Initiative, 2006. <http://dublincore.org/>
- [4] Ehrig, H., M. Pfender and H.J. Schneider. "Graph grammars: an algebraic approach." 14th Annual IEEE Symposium on Switching and Automata Theory. pp. 167-180, IEEE, 1973.
- [5] M. Wermelinger and J. L. Fiadeiro. *A graph transformation approach to software architecture reconfiguration*. Science of Computer Programming, 44(2):133155, 2002.
- [6] T. Fischer, J. Niere, L. Torunski and A. Zündorf, "Story Diagrams: A new Graph Transformation Language based on UML and Java", Proc. Theory and Application to Graph Transformations (TAGT'98), vol. 1764 of LNCS, 2000, Springer.
- [7] Hausmann, J.H., Heckel, R., Lohmann, M.: "Model-based Discovery of Web Services", In Proc. of the IEEE International Conference on Web Services (ICWS), June 6-9, 2004, USA,
- [8] Heckel, R. and L. Mariani. "Automated Conformance Testing of Web Services." In Proc. of 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005), vol. 3442 of LNCS, Springer, pp. 34-48.
- [9] D. Hirsch. *Graph transformation models for software architecture styles*. PhD thesis, Departamento de Computacion, Universidad de Buenos Aires, 2003.

- [10] J. Laprie, B. Randell, C. Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Transactions on Dependable and Secure Computing, (Vol.1, No.1) (2004) pp. 11-33
- [11] Zheng Li, Jun Han, Yan Jin, "Pattern-Based Specification and Validation of Web Services Interaction Properties", In Proc. Third International Conference on Service-Oriented Computing (ICSOC 2005), Vol. 3826 of LNCS, Springer, pp. 73-86.
- [12] Object Management Group. Model Driven Architecture. <http://www.omg.org/mda>
- [13] Reliable Asynchronous Message Profile (RAMP) Toolkit, IBM alphaworks. <http://www.alphaworks.ibm.com/tech/ramptk>
- [14] RM4GS Reference Guide, Version 1.0, FUJITSU LIMITED, Hitachi, Ltd. and NEC Corporation, 2004. <http://xml.coverpages.org/rm4gs20041125-reference.pdf>
- [15] Rodrigues,G.N., G. Roberts, W. Emmerich and J. Skene. "Reliability Support for the Model Driven Architecture." In Proc. Workshop on Software Architectures for Dependable Systems (WADS 2003), Vol. 3069 of LNCS, Springer, pp. 79-98, 2004.
- [16] Software Engineering for Service-Oriented Overlay Computers (SENSORIA) European project IST-3-016004. <http://sensoria.fast.de>
- [17] Web Services Architecture, 2004. <http://www.w3.org/TR/ws-arch/>
- [18] Web Service Modeling Ontology (WSMO), W3C Member Submission, 2005. <http://www.w3.org/Submission/2005/SUBM-WSMO-20050603/>
- [19] Web Services Reliable Messaging Protocol (WS-ReliableMessaging) BEA Systems, IBM, Microsoft Corporation, Inc, and TIBCO Software Inc., 2002.
- [20] Web Services Reliable Messaging TC WS-Reliability 1.1 Committee Draft 1.086, OASIS Open Consortium, 2004.
- [21] Call for Participation in the OASIS Web Services Reliable Exchange (WS-RX), OASIS Open Consortium, 2005. <http://xml.coverpages.org/ni2005-05-04-a.html>

Service-oriented Assurance – Comprehensive Security by Explicit Assurances

Günter Karjoth, Birgit Pfitzmann, Matthias Schunter, Michael Waidner

IBM Research, Zurich Research Laboratory
Säumerstrasse 4, 8803 Rüschlikon, Switzerland
{gka,bpf,mts,wmi}@zurich.ibm.com

September 5, 2005

Abstract. Flexibility to adapt to changing business needs is a core requirement of today's enterprises. This is addressed by decomposing business processes into services that can be provided by scalable service-oriented architectures. Service-oriented architectures enable requesters to dynamically discover and use sub-services. Today, service selection does not consider security. In this paper, we introduce the concept of Service-Oriented Assurance (SOAS), in which services articulate their offered security assurances as well as assess the security of their sub-services. Products and services with well-specified and verifiable assurances provide guarantees about their security properties. Consequently, SOAS enables discovery of sub-services with the "right" level of security. Applied to business installations, it enables enterprises to perform a well-founded security/price trade-off for the services used in their business processes.

1 Introduction

Enterprises struggle to increase their flexibility to adapt to changing business needs. Service-oriented architectures address this challenge by decomposing enterprises into loosely coupled services, which are hosted on platforms that can adapt to changing load and performance requirements. This trend is reflected by the growth of value networks, in which enterprises specialize on their core competencies and interconnect these critical services to provide a better overall service to their customers.

Whereas current research focuses on how to integrate the business processes of these value networks, security will be a major obstacle to their wide-spread adoption. Cross-enterprise security is still addressed by long-lasting trust relationships, contracts, and manual audits. Emerging service-oriented architectures and flexible usage patterns are slowly invalidating this static closed-world approach. There exists no approach that guarantees overall security while permitting the flexibility required today.

In this paper we propose a new concept called "Service-oriented Assurance (SOAS)" that enables providers to advertise their security, allows customers to monitor the actual security of a service, and provides well-defined recourse for violations of promised security features. SOAS provides a framework to express and validate assurances. An *assurance* is essentially a statement about the properties of a component or service, typically made by the producer of the component or the provider of the service. Besides

the specification of the security properties of the component, it adds a definition of how these properties are to be measured and by whom, and a recourse for the case that the promised property does not hold. *Assurance verification* is done by determining the existence or absence of the above properties. Enterprises can then link the required level of security of their IT systems and their business requirements, namely, the level of risk that the enterprise is willing to accept. In conclusion, SOAS empowers enterprises to provide security in dynamic service-oriented architectures while automatically procuring services that offer the right level of security.

This paper first presents the taxonomy concepts of SOAS, the use of SOAS for Web Services, and a basic architecture for monitoring assurances (§2). Next, it describes the actual use of SOAS (§3) and illustrates the concept of assurances by means of some example scenarios, putting particular emphasis on the separation of the assurance from the security mechanisms that achieve the assured property (§4). Finally, it discusses related work (§5) and concludes (§6).

2 Service-oriented Assurance

SOAS is a new paradigm defining security as an integral part of service-oriented architectures. It enables services to formalize and advertise their security assurances. Based on these declarations, services can address the core challenges of secure and flexible service composition:

- What are the security properties of a given service?
- How can the actual security be measured?
- What are the assumptions, failure possibilities, and dependencies of a given service?
- What evidence can be given that a service will or does indeed meet its security promise?
- Which remedies will be taken if a service does not provide the promised security?

In the remainder of this section, we outline the use of SOAS for Web Services, the taxonomy concepts of SOAS, and a basic architecture for monitoring SOAS assurances.

2.1 From Service Level Agreements to SOAS

Web Services are the preferred way of describing services in a service-oriented architecture. If a component needs a certain service, it discovers potential providers via directories and brokers, e.g., using UDDI, WSDL, and WS-Resource descriptions, and then engages with a specific service provider. In particular in cross-domain scenarios, this engagement is governed by a Service Level Agreement (SLA), e.g., expressed in WS-Agreement, which summarizes the requester's and provider's agreement on what the service is supposed to do. An SLA defines the quality of service, how and by whom that quality is measured, and what has to happen if the service quality is insufficient. Today SLAs are often implicit (in particular for services within one organization) and in most cases fairly static and pre-negotiated. But this is expected to change – in the future service providers will be selected more dynamically and hence SLAs will be more

pervasive and negotiated in real time. This negotiation will become part of the overall process and of the overall Web Services stack.

Service-oriented Assurance adds security to this picture: Before two components engage in a service, they provide each other with assurances, i.e., security guarantees, as part of the SLA negotiation process. Examples are promises to provide certain process or data isolation, to comply with a regulation, or to accept a certain risk, or also statements of identity, etc., together with arguments why these properties hold, such as certificates for Common Criteria security evaluations, hardware-based integrity statements, or identity certificates and digital signatures.

Depending on how the SLA defines the manner in which security quality is measured, components may gather evidence during operation, i.e., information that documents and maybe even proves the state of transactions or the security posture of the component. This information can be security alarms, entries in log files, authenticated messages received from other components, hardware-based integrity measurements, etc. If something goes wrong, this information becomes the basis for fault diagnosis and forensics. Once a problem has been identified, the assurances will point to the components responsible for solving the problem and for covering damages. This is particularly important in a cross-domain scenario involving different organizations, where the result may be an actual financial recourse.

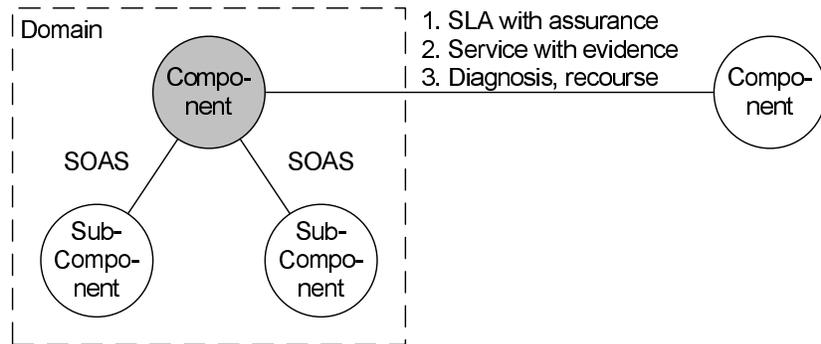


Fig. 1. Service-oriented Assurance

Figure 1 summarizes the use of assurances. We mainly consider the gray component on the left, e.g., a business process. It uses the service of another component (process), which may be in another domain, and of local sub-components. In a service-oriented architecture, there is no great difference between these two uses, except that there is by necessity a stronger dependency on some local components, e.g., the underlying operating system. In a first step, the processes negotiate an SLA with assurances. This is done in the context of the processes' own service assurances to their users. Secondly, during normal service, both processes may gather evidence of their own correct operation and of the operation of their partner; some of this evidence may be exchanged explicitly.

In case of problems, diagnosis and forensics should be possible based on the evidence gathered, and the SLA will provide procedures for a potential cross-domain recourse.

2.2 High-level Model

To enable the formalization of statements that express the security promised for a given service, SOAS defines a model as shown in Figure 2. In theoretical terms, this is in essence a meta-model of service descriptions.¹ Note that SOAS only formalizes the structure of security statements and that properties must not necessarily be expressed in a formal way; they may simply denote a certification such as a security label like EAL-4 or a privacy seal like TRUSTe having a precise meaning given from outside the SOAS model. The figure is drawn in UML, a widely used graphical design language.

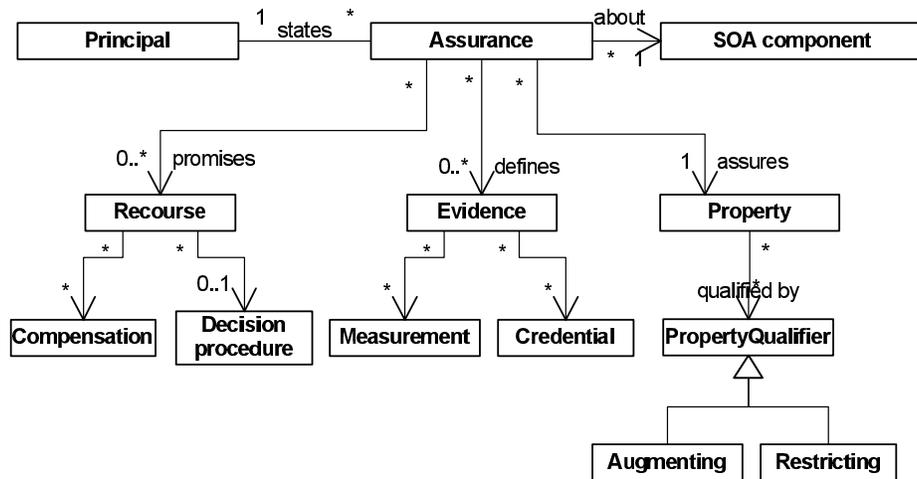


Fig. 2. Assurances on SOA components

The security of a SOA component is established by the properties it is expected to implement. Thus, declaring (security) *properties* is the foundation of the SOAS model. Properties of SOA components can be written texts (like contracts) but will often be machine-readable to enable automation and comparison beyond equality tests. Simple properties may define the I/O syntax of a service by, e.g., promising to adhere to a WSDL schema. Properties may also make statements about the actual behavior of a service stating, for example, the privacy of personal data. More examples are given in §4. This can be done using formal specifications or textual descriptions of the expected services.

¹ A meta model of SOA exists in [1], but it does not classify service descriptions or service properties. Given a meta model, corresponding models can be serialized either automatically, i.e., a “language” is defined implicitly by the UML model, or manually, e.g., into extensions of existing Web Services specification languages.

Properties either hold unconditionally or are qualified. A *qualifier* augments or restricts a given property. In an augmenting qualifier such as availability or performance, the resulting qualified property always implies the original property. Restricting qualifiers are key to modeling security. Typically they express environmental assumptions needed, e.g., a trust model specifying entities that are assumed to be correct, failure probabilities, or validation methods.

An *assurance* is a statement about a property of a *SOA component* or service, made by a *principal* such as the producer of the component or the provider of the service. Assurances define the evidence the component has to deliver to show it indeed provides the desired properties, and a specification of recourse if the component fails to provide these properties.

Evidence describes the information provided by the service to support the assurance, typically by enhancing the credibility of other elements. Mostly provided in the form of credentials, evidence may corroborate that the principal builds good components by customer references or a formal certification. Or it may corroborate the component properties, e.g., by supplying a certificate of a claimed Common Criteria evaluation or by describing the procedure used for determining the mean time between failures. Or it may corroborate a recourse, e.g., by showing that the principal has reserved funds. Evidence also defines how the property can be measured and by whom. For instance, it can be the retrieval of a log file by the service provider, a third-party audit, or a measurement signed by secure hardware included in the platform [10].

A *recourse* consists of a decision procedure and possible compensations. For dispute resolution, agreement on the interpretation of the measurement is essential to enable the parties to agree whether a property is fulfilled. A *decision procedure* provides instructions on how to deal with cases where, for example, some properties are not immediately measurable because, for instance, probabilities depend on how long one measures, or secrecy violations may not be noticed at once. If neither party fully trusts the other's measurements, the decision procedure may require that both sides measure in parallel or may even state that in case of conflicts additional proofs are provided by third parties [4].

Whenever the stated security property does not hold or can no longer be guaranteed, a *compensation* states a penalty, e.g., a sum of money, or defines a remediation process that re-instates security and is considered to be sufficient to satisfy the property. Whereas a penalty does not necessarily require the assurance to be re-established, remediation on the other hand is a well-defined process how a violation can be removed and how the system is guaranteed to reach a state that provides the given assurance. An example of the latter is the property of absence of viruses. The provider can guarantee that once a virus is discovered during a regular check, the virus is removed within 1 h and integrity of the installation is re-verified.

2.3 Monitoring Security Properties

Besides giving *explicit security assurances* as outlined above, SOAS must also support the *verifiability* of these assurances. Whereas the first challenge requires a language for specifying the assurances to be included into SLAs, verifiability of these assurances

may be achieved by providing measurements supporting the evidence in the stated properties. Figure 3 outlines possible interactions between measurement components of a SOAS-enabled service. The interactions are structured into two phases. Before actually providing a service, the provider and requester agree on the Security SLA that describes the desired security. Once an agreement is reached, the service will be provided and its security can be monitored. Depending on the trust model, an optional observer can act as a referee to decide whether the properties are indeed met.

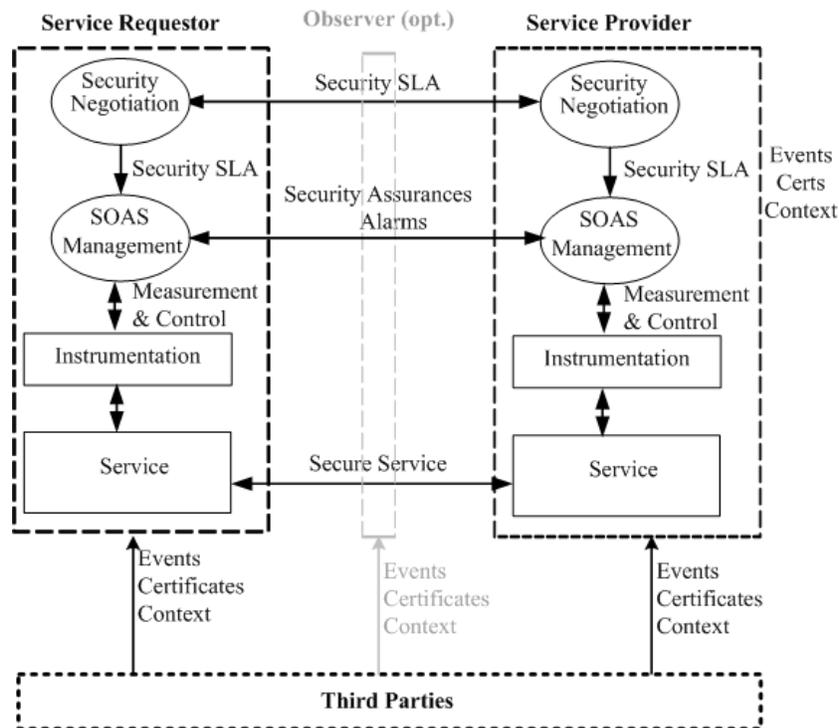


Fig. 3. Run-time monitoring of Service-oriented Assurance

To enable security monitoring, both the service and the client are instrumented. This instrumentation measures the security parameters of the service, input to a SOAS Management component verifying the given security assurance.

2.4 Types of Security Evidence

For assessing the security properties, data from various sources are needed. Both parties may evaluate system states – including security policies in place – and events that are provided through instrumentation as well as certificates, and other context from third

parties. Examples are certificates of acceptable software as well as events that represent newly discovered vulnerabilities of the installed code base.

In principle, we distinguish three main types of security measurements depending on the time period addressed:

- An *audit log* makes it possible to verify whether a system has provided the assurance in the past. Examples include log files of past virus scans, an execution history listing the executables loaded in memory [10], and access records for confidential data.
- The current *state* enables one to prove statements about a given moment in time. This can include ownership proofs about the provider implemented by certificates or the absence of a virus at this point by a virus scan.
- Convincing a verifier that certain *policies* are enforced enables a system to indicate that a security property is likely to hold in the future. Examples are training programs, access-control policies, and policies for remediating specific failures.

Depending on the trust model, additional evidence may be needed to convince the verifier that the given data is accurate. In particular for policies, it is essential to convince the verifier that they will not be replaced with inappropriate ones. A similar challenge exists for state and audit logs. If the verifier does not trust the systems providing the service, additional evidence such as audits at random times may be needed. Furthermore, evidence of insecurity may turn up in various ways, e.g., by finding confidential data on the Internet or via whistle blowers.

Properties and thus their measurements may be qualitative or quantitative. Whereas the former simply determine whether a property holds, the latter measurements determine how strong the property is. Quantitative security is not yet common-place but there are example properties such as quantitative information flow or measures like the number of known vulnerabilities and k-anonymity.

3 Applying SOAS

To gain security addressed consistently and naturally at the right places, SOAS should be integrated in the overall software architectures and tooling. However, SOAS can to some extent also be retrofitted to legacy systems by documenting and exposing their known security properties and publishing corresponding security assurances. This explicit expression of security enables service selectors to consider security as a criterion for service selection, which in turn creates a reward for security. This reward will enable appropriate economic mechanisms that lead to the highest levels of security where this is most beneficial, and allows security to increase in economically cost-effective steps.

To implement fine-grained assurance statements, security assurances must be propagated and implemented along the software stack. As a consequence, software components would either implement their own security mechanisms (e.g., a banking application using one-time passwords), use security mechanisms from lower layers (e.g., SSL encryption), or use a security infrastructure for transparent protection (e.g., anti-virus or isolation services).

Propagating explicit assurances of components requires explicit exposure of assumptions as well as a more active security management in each component. Accordingly, components need to identify their assurances based on the assurances offered by sub-components. Another important aspect is that in order to enable its use in an open environment where not all components trust each other, SOAS creates an incentive that components enable verifiability of their security properties. This means that components can produce evidence that the claimed properties are actually true. A (simple) example is ownership: A service can use attribute certificates to prove that it is owned by a certain entity. This enables higher-layer services to measure and validate the assurances provided by lower-layer services. More complex scenarios include components that guarantee to report their security status honestly based on a well-defined measurement method.

Another important application of SOAS is to use security assurances to select appropriate services and to compose services. Once sub-components declare their service guarantees, services can factor security into the decision which sub-service to select. Loosely speaking, SOAS enables a service to discover the sub-service with the right level of security and the best cost/risk trade-off. Based on the assured security properties including potential qualifiers, a service may decide which tasks to entrust to each particular service. If a sub-service does not provide the full guarantees that are needed, a service can decide to augment the guarantees, e.g., by running replicas or obtaining additional recourses that remedy the losses in case of failure.

4 Example: Security of an Outsourced Business Process

In this section we illustrate the concept of assurances based on a larger example where a bank outsources a business process to an external provider. We first identify the overall assurances and then elaborate on two specific properties.

4.1 Overall Security Agreement

The overall goal is to manage the security of a business process by identifying and verifying the security properties of its sub-processes. In practice, this initial usage of SOAS is possible without automatic verifiability of the security properties. Instead, assurance can be achieved by signed statements of the service provider containing sufficient compensation in case well-defined security measurements indicate that the promised security cannot be or has not been achieved.

Let us consider a bank that outsources a sub-process such as payment processing to an outsourcing company. As is currently done, the bank and the outsourcing provider establish a service-level agreement for the outsourced process. This SLA defines the actual service as well as key performance indicators such as availability and throughput. This SLA can be negotiated and fixed using WS-Agreement. It will also define the WSDL interfaces to the service.

Because the sub-process is critical to the business of the bank, security requirements have to be added. Using SOAS, the bank and the outsourcing provider agree upon the

actual security to be provided by the sub-process, how it is measured, and what compensation will be offered in case of failure. These security guarantees may cover different aspects of the outsourcing infrastructure and can be structured into distinct properties:

Basic integrity properties. The provider assures integrity properties on the input and the output data. In addition, it may state that input data of any kind will not lead to buffer overflows.

User management. The provider defines how users are authenticated and how access to the sub-process is restricted to the appropriate applications in the bank.

Basic infrastructure. The provider defines which availability is guaranteed and how it will be achieved, e.g., by replication, backups, and disaster recovery measures.

Isolation. The provider guarantees that this business process is completely isolated from other business. In particular, isolation holds even if processes are executed on behalf of other banks.

Application quality control. The provider defines how applications are tested and how quality is achieved. In particular, the provider guarantees that only applications that have passed a well-defined test-suite will be used to provide the service.

Security policies. The provider guarantees that the process is managed according to well-defined security policies. These policies include staff education, proper security zoning and boundary control, as well as emergency response for the corresponding services. The security policies also include virus protection and intrusion detection and response measurements.

All these properties are declared using signed statements. Because they are difficult to be verified automatically, validation can still be achieved by external auditors or audit teams from the bank or from the provider. Once the provider fails to comply, appropriate recourses from the initial assurance are used to remedy or compensate a failure.

4.2 Security Management – Customer Isolation

In an outsourcing environment, data owned by different customers must be isolated; i.e., no information may flow between customers except through well-defined business processes. This causes no problem in today's outsourcing environment, where most resources and applications are dedicated to one customer. In the case of shared applications or resources, however, they must be certified to provide appropriate isolation. As a consequence, a property promised may be that "there is no information flow between all services of this customer and any service provided to another customer".

To securely implement above guarantees, the provider either has to provide dedicated resources for each customer or to guarantee that no shared resource leaks any information between customers. This guarantee for shared applications can be done by means of an evaluation and certification. An alternative is the provision of virtualized resources (such as logical partitions) that are dedicated to each customer, enabling different customers to share one machine but still providing guaranteed separation. However, as it is hard to analyze whether a shared application allows information flow or not, both parties may have to accept some level of risk.

Depending on the trust the bank puts into the provider, the actual mechanisms that are used as well as their verifiability will differ. One way to provide assurance is to provide a signed statement of the provider or an auditor. If the trust in the provider or the auditor was unjustified, the customer may notice a violation only if the undesired information flow has visible effects, e.g., secret data clearly being used by competitors. For these cases, there must be compensation. The decision procedure may be aided by watermarking techniques. However, mechanisms where the service provider is trusted to notify someone of security violations are known and can be effective, as the experience with the California Senate Bill No. 1386 shows.²

Property qualifiers can be used to define limitations of virtualization including the requirement that certain services be not virtualized, virtualized on a dedicated resource, or hosted on machines satisfying certain criteria such as physical security protection or location [2]. An example of the latter are the concerns that Canadian personal data will fall under the US Patriot Act once they are hosted on machines that are physically in the USA [8].

4.3 Security Management – Virus Protection

Service users require that machines hosting critical services follow basic security guidelines. The property that is promised by a service is that the machine providing a service is managed according to well-known security guidelines. Such guidelines usually require sound patch management, firewalls, and appropriate virus protection.

Assurance of appropriate virus protection, for example, can be implemented in different ways. Using certification and recourse, the service provider promises to manage the machines according to the guidelines and certifies this including recourse. As virus attacks are usually quite visible by loss of availability, the bank may not require specific measurements in the assurance if the recourse is sufficient to cover potential losses. Alternatively, sound virus protection may be indicated by means of an audit trail of recent virus scans to convince a verifier that no virus activity was detected while a given service was being provided. Moreover, assurance for this property can be provided by means of integrity-based computing (IBC) mechanisms. For virus protection, IBC can prove at regular intervals that a virus scanner has been resident in memory and not been invalidated.

5 Related Work

Several models and languages formalize agreements (contracts) on electronic services [4, 11, 12], covering agreement specification as well as system architecture. However, they mainly focus on specific aspects of services. For example, WSLA is a language for the specification of quality-of-service agreements for Web services. Besides providing a type system for SLA artifacts, WSLA identifies the contractual parties, specifies the

² Summaries of incidents cataloged on PIPEDA and Canadian Privacy Law can be found at <http://www.privacylawyer.ca/blog/2005/02/summaries-of-incidents-cataloged-on.html>.

characteristics of the service and its observable parameters, and defines the guarantees and constraints that may be imposed on the SLA parameters [4].

WS-Agreement is a standardization effort defining an agreement format, an agreement establishment protocol, and a runtime agreement monitoring interface. Agreement terms represent contractual obligations, including specific guarantees given [5]. Guarantee terms specify service level objectives, a qualifying condition under which objectives are to be met, and a business value giving the importance of meeting these objectives.

The Composite Assurance Mapping Language (CAML) provides a notation for claim trees for the assurance arguments related to enterprise security objectives, providing causalities, relationships, vulnerabilities, threats, and other system- and environment-related issues [6]. A CAML specification hierarchically refines security claims about the system into sub-claims that, eventually, are linked with the evidence that a claim is satisfied. Refinement is supported by the general strategy, assumptions, and dependencies, justifying reasons, and contextual models.

Security properties of components can be measured and verified by using products such as Symantec Enterprise Security Manager or IBM Tivoli Security Compliance Manager (SCM). SCM gathers information from multiple computer systems, such as registry and application information, analyzes the data, and produces reports to reveal adherence to security policies. Collectors retrieve specific data by reading files or running an executable program. Data collected on client systems is stored in a database on the server. Conditions are expressed as SQL statements that analyze data from the database tables to provide a list of client machines violating the conditions.

Also trusted computing allows one to verify the integrity of a platform (attestation), whereby secure boot and strong isolation guarantee integrity. Remote attestation authenticates software to remote parties. However, attestation based only on the configuration of software and hardware components entails the problem of managing the multitude of possible configurations, system updates, and backups [3, 7, 9]. A trusted virtual machine, as for example proposed by Haldar et al [3], can execute platform-independent code to attest programs, thus certifying various properties of code running under it by explicitly deriving or enforcing them. SOAS assurances may provide the language to express these properties and the way they should be verified.

6 Conclusion

Service-oriented Assurance enables products and services to provide well-specified security guarantees, which can be monitored and validated. These assurances enable enterprises to select services that offer the right level of security. Our example illustrates that it is feasible to specify important security properties in a vendor-agnostic and platform-independent way. As a consequence, we believe that SOAS is the logical future of security in service-oriented architectures.

Our proposal is only a first step in this direction. Further work is required in the formalization of a broad range of specific security properties and on assurance verification as well as on service composition. There is still a long way to go before security risks are comprehensively managed and become normal economic factors on the business

layer. Nevertheless we have demonstrated a framework that shows how the objectives stated above can be achieved and that first meaningful ways exist to instantiate this framework based on current software and hardware capabilities.

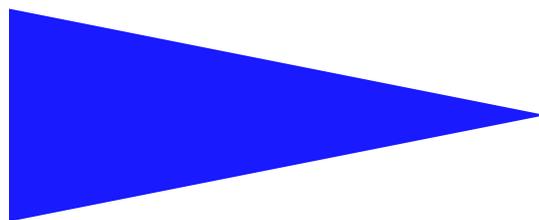
Acknowledgments

We want to thank Bob Blakley, Tom Corbi, Bruce Harreld, Linda Henry, Anthony Nadalin, Nataraj Nagaratnam, Chris O'Connor, Charles Palmer, Ronald Perez, and Andreas Wespi for interesting feedback.

References

1. L. Baresi, R. Heckel, S. Thöne, and D. Varró. An architectural style for service-oriented architectures. www.upb.de/cs/ag-engels/ag_engl/People/Thoene/MRDSA/SOA-Metamodel.pdf, Sept. 2003.
2. J. L. Griffin, T. Jaeger, R. Perez, R. Sailer, L. van Doorn, and R. Cáceres. Trusted Virtual Domains: Toward secure distributed services. In *Workshop on Hot Topics in System Dependability*, 2005.
3. V. Haldar, D. Chandra, and M. Franz. Semantic remote attestation: A virtual machine directed approach to trusted computing. In *USENIX Virtual Machine Research and Technology Symposium*, pages 29–41, 2004.
4. A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management, Special Issue on E-Business Management*, 11(1), Mar. 2003. Plenum Publishing Corporation.
5. H. Ludwig, A. Dan, and R. Kearney. Cremona: an architecture and library for creation and monitoring of WS-agreements. In *2nd International Conference on Service Oriented Computing (ICSOC '04)*, pages 65–74, New York, NY, USA, 2004. ACM Press.
6. J. S. Park, B. Montrose, and J. N. Froscher. Tools for information assurance arguments. In *DARPA Information Survivability Conference and Exposition II (DISCEX'01)*, volume 1, pages 287–296, 2001.
7. J. Poritz, M. Schunter, E.V. Herreweghen, and M. Waidner. Property attestation — scalable and privacy-friendly security assessment of peer computers. IBM Research Report RZ 3548, 2004.
8. Public Sector Outsourcing, Information & Privacy Commissioner for British Columbia. Privacy and the USA Patriot Act - Implications for British Columbia. www.oipc.bc.org/sector_public/usa_patriot_act/pdfs/report/privacy-final.pdf, Oct. 2004.
9. A.-R. Sadeghi and C. Stübke. Property-based attestation for computing platforms: Caring about policies, not mechanisms. In *New Security Paradigm Workshop 2004*, pages 67–77. ACM Press, 2005.
10. R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *11th ACM Conference on Computer and Communications Security*, pages 308–317. ACM Press, 2004.
11. J. Skene, D. Lamanna, and W. Emmerich. Precise service level agreements. In *26th Int. Conference on Software Engineering*, pages 179–188. IEEE Computer Society Press, 2004.
12. V. Tosci, B. Pagurek, and K. Patel. WSOL – a language for the formal specification of classes of service for web services. In *International Conference on Web Services (ICWS'03)*, pages 375–381. CSRA Press, 2003.

PUBLICATION
INTERNE
N° ---



A SURVEY ON COMMUNICATION PARADIGMS FOR
WIRELESS MOBILE APPLIANCES

DAMIEN MARTIN-GUILLEREZ AND MICHEL BANÂTRE
AND PAUL COUDERC

A survey on communication paradigms for wireless mobile appliances

Damien Martin-Guillerez^{*} and Michel Banâtre^{**} and Paul Couderc^{***}

Systèmes communicants
Projet ACES

Publication interne n° — — May 2006 — 14 pages

Abstract: During the design of wireless services, one must have in mind problems linked to the mobility of the devices. Indeed, when a device moves, it can lose the network, acquire it, or the performances of the network can change. To handle those issues, wireless services must be designed with the knowledge of what can happen in the network.

In this paper, we will survey existing paradigms to deal with mobility in wireless networks.

Key-words: Mobile computing, wireless communications.

(Résumé : tsvp)

^{*} ENS-Cachan/Bretagne - dmartin@irisa.fr

^{**} INRIA - banatre@irisa.fr

^{***} INRIA - pcouderc@irisa.fr

Etat de l'art des paradigmes de communication pour terminaux mobiles communicants

Résumé : Lors de la construction de services utilisant des réseaux sans-fils, la mobilité des terminaux est souvent un problème à prendre en considération. En effet, la qualité des connexions influence sur les fonctionnalités que le service peut offrir. Dans un contexte mobile, les connexions sont temporaires et leur qualité est très variable.

Dans cet état de l'art, nous présentons les différents problèmes survenant dans ce genre de réseaux ainsi que les différentes solutions existantes.

Contents

1	Introduction	3
2	Wireless network technologies	4
2.1	IEEE 802.11	4
2.2	BlueTooth	4
2.3	Other technologies	4
3	Adaptive approaches	5
3.1	Handover handling in infrastructured networks	5
3.2	Ad hoc routing	5
3.3	Predicting handover and change of cell	6
4	Ubiquitous approach	7
4.1	Spontaneous communications	8
4.2	Communication atomicity	8
4.3	Improving resource discovery	9
5	Retained solutions for the MoSAIC project	10
6	Conclusion	12

1 Introduction

Wireless interfaces usually support only short-range communications. Thus, wireless appliances can only communicate with close neighbors and messages require to be retransmitted to attain the whole network. Moreover, mobility of appliances leads to regular changes of neighbors and of network capabilities.

Several proposals have been made to address these issues. Adaptive approaches concentrate on simulating classical networks by palliating disconnection, bandwidth decrease and limited communication range using quality of service and routing mechanisms. On the contrary, ubiquitous approaches address only neighbor devices and use wireless limitations and mobility as information for the services.

In this survey, section 2 presents mainstream wireless technologies and their limitations. Section 3 presents the adaptive approaches and section 4 the ubiquitous ones. Finally, after underlining the retained solutions for the MoSAIC project [14] in section 5, we conclude in section 6.

2 Wireless network technologies

IEEE 802.11 and BlueTooth are the main communication technologies that exist today. In this section, we will present these technologies and some new ones that are starting to appear.

2.1 IEEE 802.11

IEEE 802.11 [24, 8], often nicknamed WiFi, is a standard for wireless local area networks (WLAN). It was designed to remove cables in short-range local area networks. Its communication range is about 50 meters in an open office environment for a bandwidth from 10 to 54 Mbps per channel. It has two main communication modes: infrastructure and ad hoc.

In the infrastructure mode, IEEE 802.11 uses access points to act as routers between peers (and the possible wired network). In this mode, access to other terminals is similar to Ethernet. Each wireless interface has a unique identifier (its MAC address) which is used when addressing another interface. All messages pass through the access point which redirects them to the corresponding interface (if it has registered to the access point).

In the ad hoc mode, every interface uses the same network parameters can communicate directly with the others.

2.2 BlueTooth

BlueTooth [12] is the name for the IEEE 802.15.1 [4] standard for wireless personal area networks. It has been designed for energy efficient communications in very short range (less than 10 meters). It has a 1 Mbps bandwidth. Contrary to the IEEE 802.11 ad hoc mode, BlueTooth relies on a master-slave communication paradigm where each message between two peers passes through a node called the *piconet master*¹.

To enter a network, BlueTooth uses a discovery mode (Inquiry Scan) during which it can detect new nodes but the scan requires 1,28 seconds per node minimum. Moreover, during the discovery mode, the interface cannot be reached for other activities. The very slow discovery time and the invisibility during the discovery are real problems to when it comes to handling mobility. Therefore, BlueTooth is generally used for communication between fairly static sets of BlueTooth capable appliances (like a PC and a BlueTooth mouse).

2.3 Other technologies

Several other technologies have started to appear. ZigBee (IEEE 802.15.4) is a more energy and memory efficient technology than BlueTooth for close range networks. Unfortunately, it suffers from a lower bandwidth (250 kbps) and discovery time. ZigBee is aimed mainly at networks of non-mobile sensor nodes.

WiMax or IEEE 802.16 [23] is a standard for wide area wireless networks and provides a network similar to the the infrastructure mode of IEEE 802.11. It can provide a bandwidth

¹A *piconet* is defined as a small BlueTooth network including a master and several slaves.

from 4.5 Mbps to 21 Mbps per channel in a range of 1 to 15 kilometers. Thus, it is mainly designed for a large-scale cellular service.

3 Adaptive approaches

The usual approaches to handle mobility in wireless networks are the adaptive ones. These approaches rely on implicit mechanisms to emulate a continuous connectivity in a mobile environment.

3.1 Handover handling in infrastructured networks

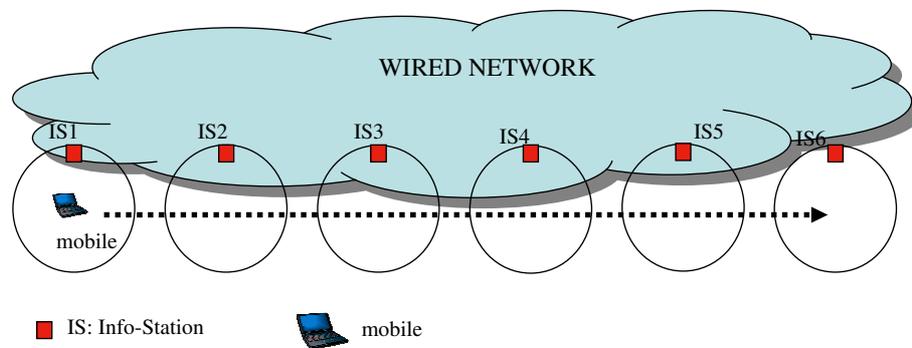


Figure 1: Infrastructured network.

Multiple linked cells (IS1 to IS6) cover the area and the mobile terminal can move between them.

In an infrastructured wireless network, each cell is covered by a wireless antenna (e.g., a HotSpot or an InfoStation [10]) like in figure 1. Mobile terminals move between those cells and thus suffer from bandwidth variations (loss of network and decrease of bandwidth). To address this issue, QoS (Quality of Service) and caching mechanisms are generally used.

QoS mechanisms decrease or increase the bit-rate of the data sent to the terminal depending on the network capabilities. For a video stream for instance, the encoding bit-rate of the video can be modified to match the network bandwidth. Caching mechanisms send data that will be of use later when the bandwidth of the network is high. In disconnected or poor bandwidth zones, the mobile terminal can work on pre-loaded data.

3.2 Ad hoc routing

In mobile ad hoc networks (MANETs) [13], each terminal can act as a router (see figure 2) like in peer to peer networks [5]. Protocols like LAR [15] or DREAM [2] use flooding like

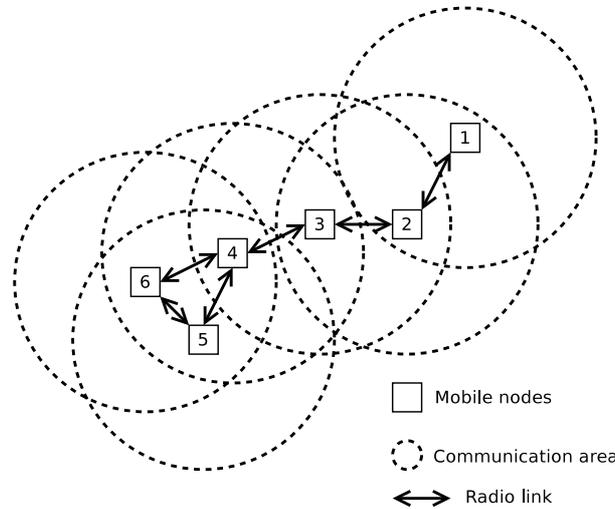


Figure 2: A Mobile Ad Hoc Network.

Each nodes can only communicate with some others (6 can only talk to 4 and 5). To talk to other nodes, messages must be re-transmitted by intermediary nodes (a message from 6 to 2 must be routed by nodes 4 and 3). The links between nodes change during the evolution of the network (nodes are mobile).

methods to construct routes. LAR sends a request to a certain area (the expected zone) when a route needs to be constructed to send a message: it is a reactive protocol. On the other hand, DREAM is proactive: it regularly reconstructs its routes by sending discovery messages in a precise zone (similar to the expected zone of LAR). The expected zones of LAR and DREAM are both computed using cinematic information although the computations are different. These *expected zones* successfully reduce the bandwidth and generally scale correctly with the mobility.

3.3 Predicting handover and change of cell

The knowledge of the way a terminal moves is a key point for handling mobility in wireless networks. If the system is able to know where and when a terminal will switch cells or will be disconnected then it can anticipate mobility by sending data to the next cell, authenticating the terminal in the next cell, etc...

Abowd et al. [1] and Narendran et al. [16] have proposed mechanisms to predict the minimum time before a terminal leaves the coverage zone. It can be done using regular measurement of the received signal power. This power is proportional to the inverse of the square of the distance ($\frac{1}{d^2}$) in short range and to $\frac{1}{d^4}$ in long range. Thus, using regular

acquisitions of the received signal power, one can compute the variation of the distance and the probable time when the terminal will leave the cell. In practice, we only estimate if the signal power will be high enough during the transmission time using its measured variation. Furthermore, the IEEE 802.11F [24] standard proposes with the Inter Access Point Protocol to learn paths between different cells to predict where to send the data after the disconnection from the first cell. When a terminal moves from a cell to another, a new path is created between those two cells and data for terminals will be sent to the second cell when they start to leave the first one.

4 Ubiquitous approach

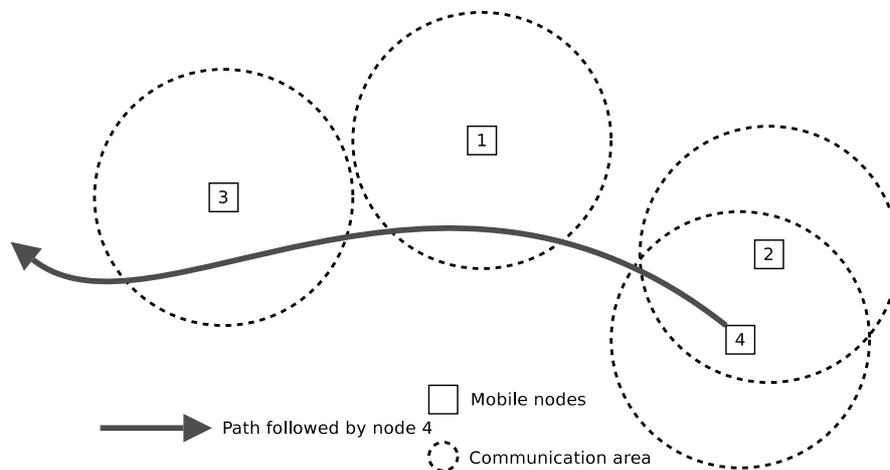


Figure 3: A ubiquitous service.

Each node provides information in its range of communication. Node 4 can currently access to the information of node 2 and will be able to access to the information of node 1 and 3 during its trip. The accessible information is related to its position.

Contrary to adaptive approaches, the ubiquitous approach based on the Mark Weiser's vision [22] considers that each wireless terminal is an information system that has to be available only in close range (figure 3). For example, electronic tags provide informations on each painting of a museum and a visitor's handled computer can read those tags and so act as a virtual guide.

4.1 Spontaneous communications

The underlying concept in ubiquitous computing is spontaneous communications. As ubiquitous computing's main principle is that of invisible embedded computers that enable users to seamlessly interact with a dynamic environment. For example, CoolTown [20] associates everyday objects with wireless appliances that contain information in the HTML format. They beacon identifiers to mobile terminals. The terminal can then display the related HTML information automatically when passing near the object. In this system, locality is a way to address information.

SPREAD [6] is a middleware for ubiquitous applications. It interprets the physical space as an addressing space. Each terminal can provide information in a tuple form and accesses information by selecting tuples of a certain form (like in Linda [11]). Those tuples are accessible only to terminals in communication range. Thus, a mobile terminal can access only neighbor information. An example of an application built using SPREAD is UbiBus, which is aimed at helping visually-impaired persons. A bus equipped with a wireless appliance using UbiBus spreads a tuple indicating the line number. User appliances that are close to the bus can acquire this information. The user is then alerted when the bus arrives.

Persend [21] is another system that uses physical space as a parameter for the addressing mechanism. It proposes to establish continuous database requests that are linked to locality. Consider the example of a terminal B that publishes a list of music albums he wants to sell and a terminal A that wants to know the list of music albums for less than ten euros. A's list evolves during the time: if A goes near B then he gets the list of B's albums that are for sale for less than ten euros, if B changes the price of an album, the request is modified accordingly and if A leaves B neighborhood, then B's albums are removed from A's list.

4.2 Communication atomicity

For some applications like taxi reservation, there is a need for transactions (e.g., atomicity of communications). Unfortunately, in presence of data loss and disconnection, the atomicity of a transaction cannot be guaranteed. However, Pauty et al. [17] have proposed a protocol to approach atomicity in the context of spontaneous communications.

They propose to add a *take* operation in SPREAD which removes a tuple from available ones. This operation is based on a four-way handshake to acknowledge the transaction as shown in figure 4. This handshake can only be started in a restricted area that can be determined either by using GPS or the strength of the radio signal used to communicate. This restricted area is calculated so that the communication time will be enough for the *take* operation to be completed as shown in figure 5.

Such a mechanism reduces the number of failures during atomic operations in spontaneous communications almost to null if the restricted area is small enough. The size of the ideal restricted area can be easily computed using the effective bandwidth, and the speed of change of signal (which depends on the speed of the user).

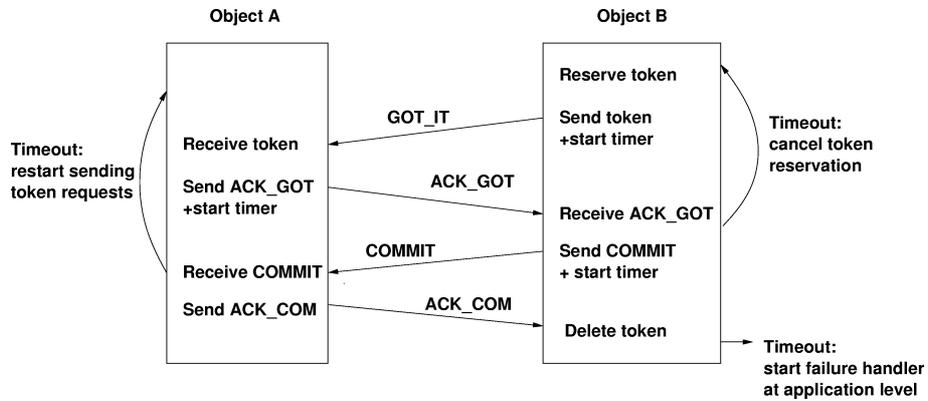


Figure 4: The *take* operation handshake.

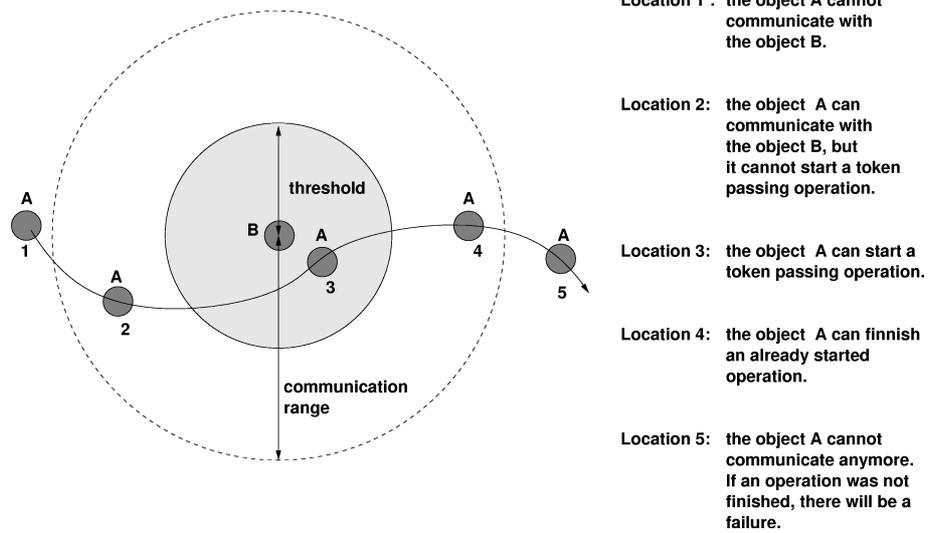


Figure 5: A geometric constraint guarantees a minimum communication time for the *take* operation.

4.3 Improving resource discovery

Discovery of hosts and resources can be quite long as seen in section 2.2. In the context of high mobility, this discovery time can be a heavy burden for spontaneous communication

applications. Le Bourdon et al. [3] have proposed to create *spontaneous HotSpots* that will register nearby terminals and their resources and propagate the information.

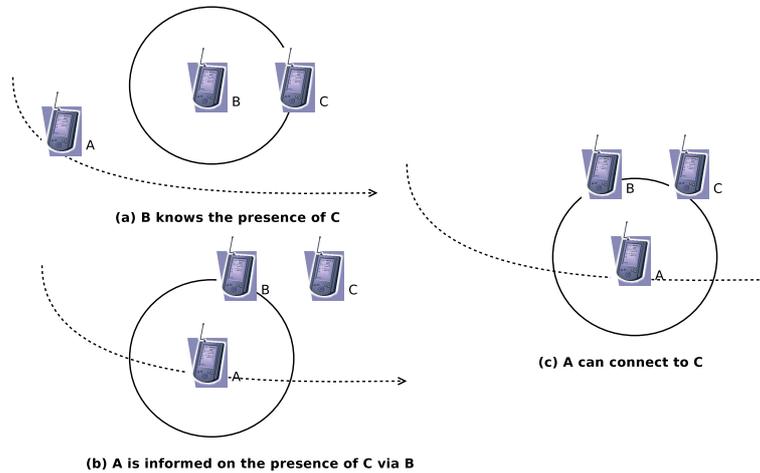


Figure 6: Propagation of resources in *spontaneous HotSpots*.

Figure 6 shows how a terminal B can transmit informations about its neighbor to a terminal A. The terminal A can then detect C without using the discovery mode of the Bluetooth interface. Of course, resources proposed by terminal C can also be given to terminal A so the latter does not have to discover them by asking C when they meet but may use them directly

Others solutions exist to improve the discovery mode of the Bluetooth protocol but they rely on other hardware like IrDA, visual tags or dedicated devices.

5 Retained solutions for the MoSAIC project

The MoSAIC project is aiming at creating a collaborative backup for mobile appliances. Therefore, it needs a network layer providing spontaneous communications. The IEEE 802.11 standard was chosen for wireless communications because of its availability, bandwidth and communication range. The general design of MoSAIC given by Courtes et al. [7] asks for two mechanisms: a discovery and a transmission mechanisms. The figure 7 shows the required interactions for the network layer.

First, a discovery mechanism is required. It should regularly broadcast a beacon message to neighbor terminals. A neighbor terminal receiving this beacon can then acknowledge his availability for backup. The discovery mechanism receiving an acknowledgement to the beacon will signal to the other layers of MoSAIC that there is a terminal ready to save

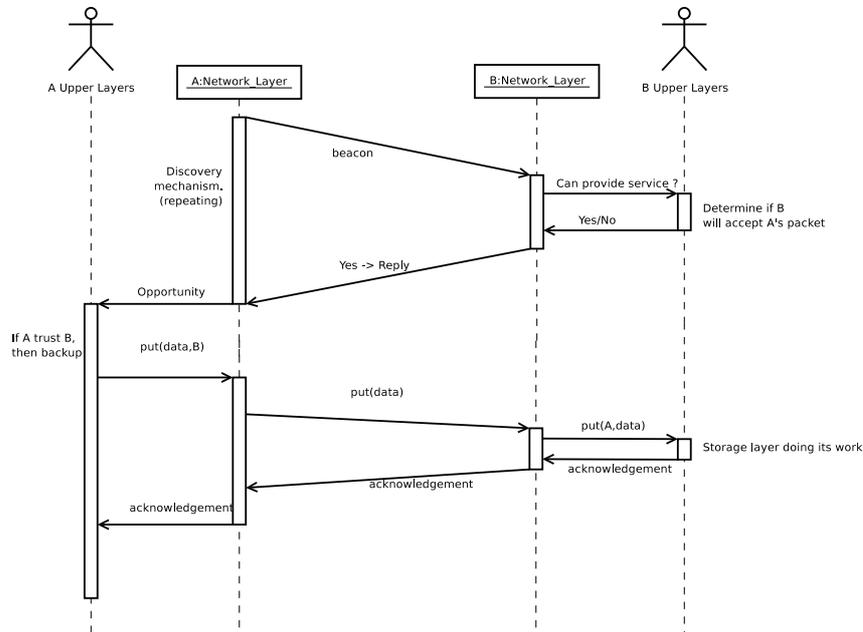


Figure 7: Interactions of the network layer in the MoSAIC project.
A terminal A tries to backup its data and a terminal B accept to save A's data.

our data. The rate of sending the beacon will change depending on the amount of data to backup. Of course, the sending of the beacons will stop when all data have been backed-up. Moreover, the wireless interface can stop listening when there is no more space available for backups to avoid waste of energy. This discovery is similar to SPREAD discovery and can be done using MAC address with IPv6 [9] and UDP [18] broadcast messages.

Second, a transmission mechanism is required. The other layers of MoSAIC ask the network layer to put its data on a recently seen terminal. The network layer initiate a transmission that need to be the more atomic we can. This transmission can be done using a handshake and restricted zone like proposed by Pauty et al. However, MoSAIC only need to know if the data was correctly transmitted. Thus, TCP [19] communications can handle the required acknowledgements. The restricted zone can be determined by the power signal. Unfortunately, most IEEE 802.11 cards do not support per link statistics but give you an evaluation of the link quality of recent transmissions. So, the restricted zone will be a lower bound for the received signal power; under that bound, transmissions will be refused to avoid unnecessary energy consumption.

6 Conclusion

In this survey, we have presented the two main wireless technologies used today and their characteristics in section 2. We have seen that IEEE 802.11 is a good middle-range wireless protocol but suffers from excessive power consumption compared to Bluetooth. On the other hand, Bluetooth has a very slow discovery protocol which is a big disadvantage for spontaneous communications.

Communication paradigms for wireless mobile appliances can be divided between the adaptive and the ubiquitous approaches. Adaptive approaches try to reduce the disadvantages of mobility by several mechanisms like caching and QoS for infrastructured networks and restricting discovery to expected zones in MANETs. Adaptive approaches also use prediction techniques using power consumption and path learning to improve data distribution or discovery. On the contrary, ubiquitous approaches use locality as a means to address data. CoolTown, SPREAD and Persend use the communication range as a way to know if the information will be useful to the user. We have also seen ways to improve spontaneous communications with atomic transactions in section 4.2 and discovery with *spontaneous hotspots* in section 4.3. Finally, we have seen several mechanisms applied to the specific case of the MoSAIC collaborative backup in section 5.

References

- [1] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: a Mobile Context-Aware Tour Guide. *ACM Wireless Networks*, 3(5):421–433, Oct. 1997.
- [2] S. Basagni, I. Chlamtac, R. V. Syrotiuk, and B. A. Woodward. A Distance Routing Effect Algorithm for Mobility. In *the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 49–64, Oct. 1998.
- [3] X. L. Bourdon, P. Couderc, and M. Banâtre. Spontaneous Hotspots: Sharing Context-Dependant Resources Using Bluetooth. In *Self-adaptability and self-management of context-aware systems (SELF'06)*, July 2006.
- [4] 802.15 Specifications for wireless personal area networks. IEEE Standards. 802.15.1 is known as Bluetooth, 802.15.4 is known as ZigBee.
- [5] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *The Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.
- [6] P. Couderc and M. Banâtre. Ambient computing applications: an experience with the SPREAD approach. In *the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, pages 291–299, Jan. 2003.

- [7] L. Courtès, M.-O. Killijian, and D. Powell. Storage Tradeoffs in a Collaborative Backup Service for Mobile Devices. Technical Report 05673, LAAS-CNRS, Apr. 2006.
- [8] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai. IEEE 802.11 Wireless Local Area Network. *IEEE Communications Magazine*, 35(9):116–126, Sept. 1997.
- [9] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) - Specification. RFC 2460, The Internet Society, Dec. 1998.
- [10] R. H. Frenkiel, B. R. Badrinath, J. Borràs, and R. D. Yates. The Infostations Challenge: Balancing Cost and Ubiquity in Delivering Wireless Data. *IEEE Personal Communications*, 7(2):66–71, Apr. 2000.
- [11] D. Gelernter. Generating Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, Jan. 1985.
- [12] J. Haartsen, M. Naghshineh, J. Inouye, O. Joeressen, and W. Allen. Bluetooth: Vision, goals, and architecture. *Mobile Computing and Communications Review*, 2(4):38–45, Oct. 1998.
- [13] D. Johnson. Routing in Ad Hoc Networks of Mobile Hosts. In *The IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, Dec. 1994.
- [14] M.-O. Killijian, D. Powell, M. Banâtre, P. Couderc, and Y. Roudier. Collaborative Backup for Dependable Mobile Applications. In *The 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (Middleware)*. ACM, Oct. 2004.
- [15] Y.-B. Ko and N. H. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. *Wireless Networks*, 6(4):307–321, July 2000.
- [16] B. Narendran, P. Agrawal, and D. Anvekar. Minimizing cellular handover failures without channel utilization loss. In *the Global Telecommunications Conference (GLOBECOM'94)*, volume 3, pages 1679–1685, Dec. 1994.
- [17] J. Pauty, P. Couderc, and M. Banâtre. Atomic token passing in the context of spontaneous communications. Technical Report 5445, IRISA/INRIA Rennes, Jan. 2005.
- [18] J. Postel. User Datagram Protocol. RFC 768, The Internet Society, Aug. 1980.
- [19] J. Postel. Transmission Control Protocol. RFC 793, The Internet Society, Sept. 1981.
- [20] S. Pradhan, C. Brignone, J.-H. Cui, A. McReynolds, and M. T. Smith. Websigns: Hyperlinking Physical Locations to the Web. *IEEE Computer Magazine*, 34(8):42–48, Aug. 2001.
- [21] D. Touzet, F. Weis, and M. Banâtre. PERSEND: Enabling Continuous Queries in Proximate Environments. In *the Workshop on Mobile and Ubiquitous Information Access*, Sept. 2003.

- [22] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, pages 94–10, Sept. 1991.
- [23] 802.11 Specifications for broadband wireless access. IEEE Standards. Known as WiMax.
- [24] 802.11 Specifications for wireless local area networks. IEEE Standards. Known as WiFi.



University of Newcastle upon Tyne

COMPUTING SCIENCE

Secure and Provable Service Support for Human-Intensive Real-Estate Processes

E. Ribeiro de Mello, S. Parastatidis, P. Reinecke, C. Smith, A. van Moorsel, J. Webber.

[This work won **FIRST PRIZE** in the **IEEE International Services Computing Contest**, September 2006].

TECHNICAL REPORT SERIES

No. CS-TR-960 May, 2006

Secure and Provable Service Support for Human-Intensive Real-Estate Processes

Emerson Ribeiro de Mello, Savas Parastatidis, Philipp Reinecke, Chris Smith, Aad van Moorsel, Jim Webber.

Abstract

We introduce SOAR, a service-oriented architecture for the real-estate industry that embeds trust and security, allows for formal correctness proofs of service interactions, and systematically addresses human interaction capabilities through web-based user access to services. We demonstrate the features of SOAR through a Deal-Maker service that helps buyers and sellers semi-automate the various steps in a real-estate transaction. This service is a composed service, with message-based interactions specified in SSDL, the SOAP service description language. The implemented embedded trust and security solution deals with the usual privacy and authorization issues, but also establishes trust in ownership and other claims of participants. We also demonstrate how formal techniques can proof correctness of the service interaction protocol specified in SSDL. From an implementation perspective, a main new contribution is a protocol engine for SSDL. A proof-of-concept demonstration is accessible for try-out

Bibliographical details

RIBEIRO DE MELLO, E., PARASTATIDIS, S., REINECKE, P., SMITH, C., VAN MOORSEL, A., WEBBER, J..

Secure and Provable Service Support for Human-Intensive Real-Estate Processes

[By] Emerson Ribeiro de Mello, Savas Parastatidis, Philipp Reinecke, Chris Smith, Aad van Moorsel, Jim Webber.

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2006.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-960)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE

Computing Science. Technical Report Series. CS-TR-960

Abstract

We introduce SOAR, a service-oriented architecture for the real-estate industry that embeds trust and security, allows for formal correctness proofs of service interactions, and systematically addresses human interaction capabilities through web-based user access to services. We demonstrate the features of SOAR through a Deal-Maker service that helps buyers and sellers semi-automate the various steps in a real-estate transaction. This service is a composed service, with message-based interactions specified in SSDL, the SOAP service description language. The implemented embedded trust and security solution deals with the usual privacy and authorization issues, but also establishes trust in ownership and other claims of participants. We also demonstrate how formal techniques can proof correctness of the service interaction protocol specified in SSDL. From an implementation perspective, a main new contribution is a protocol engine for SSDL. A proof-of-concept demonstration is accessible for try-out

About the author

Emerson Ribeiro de Mello is with Federal University of Santa Catarina, Departamento de Automacao e Sistemas, Florianopolis, Brasil.

Savas Parastatidis is with Microsoft, Redmond, USA..

Philipp Reinecke is with Humboldt-Universitaet Berlin, Institut fuer Informatik, Berlin, Germany,

Christopher Smith is a PhD student at the University of Newcastle-upon Tyne, and researching in the area of [self-managing systems](#); specifically software architectures and distributed decision-making algorithms.

Aad van Moorsel joined the University of Newcastle in 2004. He has worked in a variety of areas, from performance modelling to systems management, web services and grid computing. Most recently, he was responsible for HP's research in web and grid services, and worked on the software strategy of the company. His research agenda is in the area of [self-managing systems](#).

Jim Webber is with ThoughtWorks, Sydney, Australia.

Suggested keywords

WEB SERVICES,
SERVICE-ORIENTED COMPUTING,
SSDL,
PI-CALCULUS,
SECURITY

Secure and Provable Service Support for Human-Intensive Real-Estate Processes

Emerson Ribeiro de Mello,^{*} Savas Parastatidis,[†] Philipp Reinecke,[‡]
Chris Smith,[§] Aad van Moorsel, Jim Webber[¶]

University of Newcastle
School of Computing Science
Newcastle upon Tyne
United Kingdom

Preface: this is a submission to the IEEE Services Computing Contest 2006. The first nine pages constitute the ‘paper’, while the overall document (that is, including the appendices) is the ‘report’. The third element of our submission is the web site to our real-estate DealMaker service, which is at <http://vs-soc.ncl.ac.uk:8180/CloseTheDeal/index.html> (account: close, password: thedeal [4]).

Abstract

The real-estate industry is an interesting target for service-oriented computing, for several reasons. The participating parties are extremely diverse and there is a high proportion of human activity and interaction involved in traditional real-estate transactions. This implies that (partial) automation of such processes must be done in highly flexible and trusted manner, with natural inclusion of the human element. The most promising response computer science offers to these challenges is found in service-oriented approaches. In this paper, we ar-

gue how service-oriented computing can potentially disrupt the real-estate industry from a business perspective. We introduce SOAR, a service-oriented architecture for the real-estate industry that embeds trust and security, allows for formal correctness proofs of service interactions, and systematically addresses human interaction capabilities through web-based user access to services. We demonstrate the features of SOAR through a DealMaker service that helps buyers and sellers semi-automate the various steps in a real-estate transaction. This service is a composed service, with message-based interactions specified in SSDL, the SOAP service description language. The implemented embedded trust and security solution deals with the usual privacy and authorization issues, but also establishes trust in ownership and other claims of participants. We also demonstrate how formal techniques can proof correctness of the service interaction protocol specified in SSDL. From an implementation perspective, a main new contribution is a protocol engine for SSDL. A proof-of-concept demonstration is accessible for try-out [4].

* Emerson Ribeiro de Mello is with Federal University of Santa Catarina, Departamento de Automacao e Sistemas, Florianopolis, Brasil, emerson@das.ufsc.br.

† Savas Parastatidis is with Microsoft, Redmond, USA, savas@parastatidis.name.

‡ Philipp Reinecke is with Humboldt-Universität Berlin, Institut für Informatik, Berlin, Germany, preinecke@informatik.hu-berlin.de.

§ Chris Smith and Aad van Moorsel are with University of Newcastle, School of Computing Science, Newcastle upon Tyne, UK, {c.j.smith4, aad.vanmoorsel}@newcastle.ac.uk.

¶ Jim Webber is with ThoughtWorks, Sydney, Australia, jim@webber.name.

1. Introduction

The real-estate industry is slowly but surely moving towards Internet-based solutions to support various aspects of their business. The currently pursued approaches (e.g., [16]) utilise web sites to make it easier to share and discover information about properties for sale, or mortgage rates offered. In addition, XML-based standards are emerg-

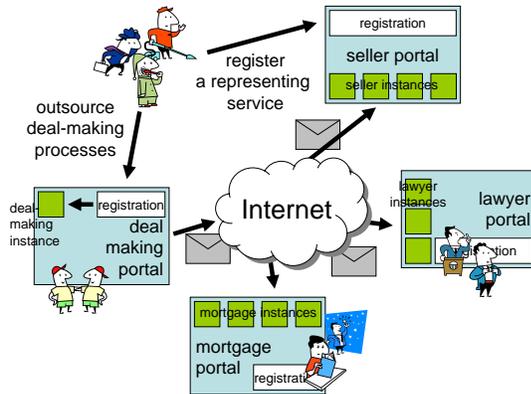


Figure 1. SOAR services landscape.

ing [9, 12, 15, 17] that support the interaction between various players (and the software packages they use), including real estate agents and mortgage lenders (see for more details Appendix A.2). Although these are good initial steps, the nature of real estate business is such that it could benefit in novel and interesting ways from more advanced service-oriented approaches to business-to-consumer and business-to-business interactions.

The objective of our work is to demonstrate how businesses and individuals can rapidly create profitable real-estate Internet services that are provably secure and correct. To that end we introduce SOAR, a Service-Oriented Architecture for the Real-estate industry. Figure 1 depicts SOAR at a high level. The main idea is that all participants in various transactions are represented by services: seller services, lawyer services, buyer services, surveyor services, etc. Services can be accessed by non-expert users through web pages for creation, configuration and termination. A service portal creates service instances when requested by users, and hosts these instances. New services can then be introduced by defining service interaction protocols in the SOAP Service Description Language and the resulting composed service can be model-checked against various liveness and deadlock properties, and has embedded a trust and security solution to assure privacy, identity and validity of user claims.

This paper describes our work during the period of the contest, from conception of the business case, to design of SOAR and the implementation of the DealMaker service. The following items are our main contributions:

- we created a business case for service-oriented computing for the real-estate industry, both for SOAR portals in general and for the DealMaker service in particular. We also argue for a possible role of standardisation bodies to successfully introduce SOAR in the diverse real-estate industry (Section 2 and Appendix A).

- we designed the SOAR architecture, with each service configurable through a web site, and personalised service instances hosted by a service provider, see Section 2.
- we suggested, designed and implemented a potential service supported by SOAR through the DealMaker service (Section 2.1).
- we embedded a security solution within SOAR to guarantee privacy, identity and validate user claims (Section 3).
- we proved correctness (with respect to the absence of starvation and race conditions) of the DealMaker service using the sequence constraints approach to protocol specification in SSDL described in Section 4.
- we implemented the DealMaker services (Section 5) and made it accessible through a demonstration web site (Appendix C and [4]).
- we designed and implemented an important new tool for the use of SSDL in managing service interaction protocols, namely an SSDL protocol execution engine (Section 5.2).

Finally, the appendices provide more details about the topics listed above, in particular with respect to the business case and the web site, and adds some reflections to our contest participation.

2. SOAR Basic Architecture

In this section we describe the main features of SOAR: basic service design, service hosting portals and personalised service instances. We also introduce the DealMaker service. First, we provide the following definitions used throughout the paper:

- *service instance* (also just *service*), see Figure 2: a run-time accessible service representation adhering to the *abstract service definition* of a particular *service type*. Service instances contain accessible *service properties*, which are stored as name-value pairs. Our security solution will provide access control at the property level.
- *service instance creation* (also just *service creation*): a *service provider* allows users to *create* (and subsequently parameterise and terminate) service instances, for the service types the provider supports. (One can think of this kind of service instance creation as the service equivalent of 'myYahoo' etc.)
- *participant*: any party involved in the system, such as lawyers, surveyors, buyers and sellers, etc., as well as the logical service representation of these parties

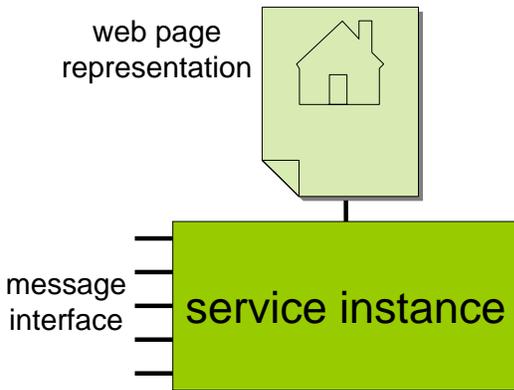


Figure 2. Service: message interfaces and web page representation.

within the system. In addition to participants, *activities* can also be represented by a service instance—example activities are drafting a contract or setting up a meeting.

In SOAR, every participant is represented by a service instance. This service contains data about the participant, and presents a messaging interface definition. The message interface allows the data to be accessed, but also allows more advanced interactions, such as ordering or stepping through stages of a workflow. The specifics of the interface definition are different for each service, and adhere to the abstract services definition for the particular participant type. Newly introduced composed services follow the same architectural design as the core services representing participants (depicted in Figure 2). That is, personalised service instances can be created and there is web page based user access to the service instances. To avoid inputting large amounts of redundant data, each service can decide to accept parameterisation referring to other service instances. For instance, a DealMaker service can be used by a buyer to create an instance that is parameterised with the service instances representing the buyer’s lawyer, etc.

Every service instance contains a web page representing the service instance, and a set of message interfaces specified in SSDL (in the implementation this is translated into WSDL documents, see Section 5). The service instance executes within a run-time environment—by default, we assume that the service instances are hosted by the service provider. We imagine service providers for sellers, buyers, lawyers, etc., or combinations thereof, see Figure 1. Alternatively, participants host their own service instances, which adhere to the message interface definition for the particular abstract service.

Figure 1 gives an idea about the landscape of real-estate services we envision. We envision portals to emerge for var-

ious participant types, for instance for lawyers, mortgage companies, buyers, sellers, etc. It is very well possible that one portal supports more than one participant type. For instance, one can imagine a portal where sellers as well as buyers register. As we discuss from the business angle in Appendix A, the portal plays a key role in bootstrapping the SOAR landscape. Participants register with their respective portals, and the portals create service instances for the registrants. In Figure 1, we therefore include the box registration, which not only indicates an opportunity to register, but also implies the ensuing process of service instance creation. The portal also provides the run-time environment to host the service instances, as indicated by the instance boxes at the various portals.

There is a number of services one can think of that exploits SOAR. In Appendix A.1 we discuss them in increasing order of complexity. There we also discuss the business case behind such services as well as behind SOAR itself.

2.1. The DealMaker Service

The DealMaker service is a complex service that demonstrates the abilities of SSDL, its associated formal proof system, and our security model. The DealMaker service helps customers to go through the steps involved in buying and selling real-estate. We have taken the process example from [7]. The service can be instantiated by any party, but for the sake of this explanation, we assume the buyer initiated the creation of a DealMaker service instance. At initialisation, it will be parameterised with the necessary information about parties involved in the deal making, such as lawyers, mortgage providers, surveyors, etc. Then, it goes through the process steps. To get an idea about the operation of the DealMaker service, it is probably simplest to read the SSDL specification of the DealMaker service given in Figure 3. The main protocol, named `Buy_Sell_Protocol`, contains a sequence of steps, each referring to another protocol: organising the mortgage, organise property viewing, add lawyer information, price negotiation protocol, etc. The stages corresponding to valuation and surveying can be executed in parallel, as one can see in Figure 3. At the end of the process, the contract gets exchanged.

We note that the DealMaker service does not attempt to *completely* automate stages of a business process. On the contrary, the assumption is that the human stays involved at all time, and many of the individual protocol steps given in Figure 3 contain status update messages sent to the right parties at the right time to assure completion of the overall process. The human then has to act on these messages for the `Buy_Sell_Protocol` to continue, and ultimately complete. In Figure 4 we display the details of the mortgage organisation protocol, as SSDL specification. It has two participants involved, the buyer and the mortgage lender. When

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <ssdl:protocol targetNamespace="http://www.ncl.ac.uk/DealMakingService/ContractExchange/protocol"
   xmlns:msgs="http://www.ncl.ac.uk/DealMakingService/ContractExchange/messages" xmlns:sc="
   urn:ssdl:protocol:sc" xmlns:ssdl="urn:ssdl:v1">
3   <sc:sc>
4     <!--Message Exchange Protocol-->
5     <sc:protocol name="Buy_Sell_Protocol">
6       <sc:sequence>
7         <sc:protocolref ref="MortgageOrganiseProtocol"></sc:protocolref>
8         <sc:protocolref ref="ViewingOrganizeProtocol"></sc:protocolref>
9         <sc:protocolref ref="LawyerRegisterProtocol"></sc:protocolref>
10        <sc:protocolref ref="SearchProtocol"></sc:protocolref>
11        <sc:protocolref ref="PriceNegotiationProtocol"></sc:protocolref>
12        <sc:parallel>
13          <sc:protocolref ref="ValuationProtocol"></sc:protocolref>
14          <sc:protocolref ref="SurveyProtocol"></sc:protocolref>
15        </sc:parallel>
16        <sc:protocolref ref="LifeAssuranceProtocol"></sc:protocolref>
17        <sc:protocolref ref="MortgageConfirmationProtocol"></sc:protocolref>
18        <sc:protocolref ref="ContractExchangeProtocol"></sc:protocolref>
19      </sc:sequence>
20    </sc:protocol>
21  </sc:sc>
22 </ssdl:protocol>

```

Figure 3. SSDL specification of the protocol followed by the DealMaker service.

the seller initiates the creation of a DealMaker service instance, it parameterises the service instance by providing buyer and lawyer information. Importantly, it does not just provide a name, but a reference to the service representing the buyer and lawyer.

The service provider that hosts DealMaker services manages the interaction given in the SSDL specification of the DealMaker service. To that end, an SSDL protocol engine runs at the service provider. It tracks how far the process is along, and initiated next steps as appropriate. The SSDL protocol execution engine is further discussed in Section 5.2.

3. Trust and Security Architecture

The SOAR architecture requires solutions for common security issues such as authentication, privacy, etc., which we discuss this in Section 3.1. However, of more specific interest to SOAR is the issue of achieving trust about claims of unknown participants in a transaction, such as about home ownership or professional credentials. We designed a SAML-based trust solution for participant claims, which we discuss in Section 3.2.

3.1. Authorization, Confidentiality and Integrity

The communication among services and between services and web users is done using SSL [6], providing basic security properties such as confidentiality and integrity. The assumption is that all service providers have acquired X.509 certificates [8], issued by a valid CA. However, SSL alone is not sufficient for identification, authentication and

authorization within services instances. Therefore, our security model uses SAML assertions [10] to provide identity as well as authenticity in message exchanges. The authorization is done by role-based access control [5] mechanism, where “roles” and “rights” are provided through SAML attribute assertions. With SAML we establish a standardized way to share credentials and an easy way to include new services or users into the system.

Service instances may have various properties that need to be protected. For instance, a seller may only be willing to share information about his/her lawyer with participants that are trying to close a deal, i.e., with services that are in same DealMaker service instance. Hence, when a new DealMaker service instance is created, each participant of this instance will receive a SAML attribute assertion (a role, e.g., `dmi:ID648s5e2:participant`), indicating that they are allowed to access “protected properties”. The default access control policy defines restrictions to some service properties, and this policy is then updated to reflect new service instances. For illustration, Figure 5 presents a small piece of our access control policy.

We also want to be able to hide the identity of the ‘real person’ that is behind a SOAR participant. In our model, the real identity of a person will be known only by the particular portal the service is created with. To other participants, a person’s identity will always be obfuscated by referring to the person through a service identifier.

3.2. Trusted Claims

In SOAR, individual participants could make unsubstantiated claims about ownership of properties, etc. In real life

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <ssdl:protocol targetNamespace="http://www.ncl.ac.uk/DealMakingService/MortgageOrganise/protocol"
   xmlns:msgs="http://www.ncl.ac.uk/DealMakingService/MortgageOrganise/messages" xmlns:sc="
   urn:ssdl:protocol:sc">
3   <sc:sc>
4     <!--Parties In Mortgage Organise Protocol-->
5     <sc:participant name="Buyer" />
6     <sc:participant name="MortgageLender" />
7     <!--Message Exchange Protocol-->
8     <sc:protocol name="MortgageOrganiseProtocol">
9       <sc:sequence>
10        <sc:choice>
11          <sc:sequence>
12            <ssdl:msgref ref="msgs:MortgageRequestSubmission" direction="in" sc:participant="
13              Buyer" />
14            <ssdl:msgref ref="msgs:MortgageRequestTemplate" direction="out" sc:participant="
15              MortgageLender" />
16            <ssdl:msgref ref="msgs:MortgageRequestCompletedTemplate" direction="in"
17              sc:participant="Buyer" />
18          <sc:choice>
19            <sc:sequence>
20              <ssdl:msgref ref="msgs:MortgageRequestAccepted" direction="out" sc:participant="
21                MortgageLender" />
22            </sc:sequence>
23            <sc:sequence>
24              <ssdl:msgref ref="msgs:MortgageRequestRejected" direction="out" sc:participant="
25                MortgageLender" />
26            </sc:sequence>
27          </sc:choice>
28        </sc:sequence>
29      </sc:choice>
30    </sc:sequence>
31    <sc:nothing />
32  </sc:choice>
33 </sc:sequence>
34 </sc:protocol>
35 </sc:sc>
36 </ssdl:protocol>

```

Figure 4. SSDL specification of the protocol followed in the mortgage organisation step.

```

1 <policy>
2   <resource id="lawyer" defaultAction="deny">
3     <allow>
4       <role id="dmi:ID648s5e2:participant" />
5       <role id="dmi:ID24n256s:participant" />
6     </allow>
7   </resource>
8 </policy>

```

Figure 5. Access control policy.

we can often easily enhance trust in such claims (such as ownership of a house) by paying a personal visit or search government archives to check if the supplied claim is true or not. However, in the virtual world of SOAR, services are often not in a position to make judgement calls about a claim of a participant is valid, possibly simply because no humans are available with the right expertise. To protect SOAR from illegitimate entrees, we use a Trusted Third Party (TTP) that is able to corroborate the claim of a participant. We can think for instance of a government institution being able to issue *claim tokens* that substantiate the claims about the ownership of a real-estate property made by a particular seller.

For example, before the creation of a service instance that offers a house for sale to all SOAR participants, the seller needs to supply house details and a claim token issued by some claim-issuing institution, indicating that the house details can be trusted. Let us assume that the seller goes in person to a government institution to show a “legal document” indicating ownership of his/her house. The government then gives the seller a claim token that the seller can forward to other SOAR participants, who then can check the validity of the claim token at the claim issuer web service.

In the demo we apply the idea of claim tokens to properties associated with a potential buyer that chooses to make use of the DealMaker service. Our implementation, based

on SAML assertions, provides a flexible and user-friendly way for participants to either obtain or check claim tokens. We think that trusted claims provide a level of trust throughout the SOAR architecture that may greatly enhance the willingness of participants to carry out business interactions through SOAR services.

4. SSDL and Formal Correctness Proof

The DealMaker service constitutes of a particularly complex orchestration of service interactions. The complex nature of the interactions makes one question the correctness of the overall process. In order to validate the correctness, we derive a π -calculus specification from the SSDL specification, and validate the resulting model formally. The way this can be done has been described in [13], and we briefly summarise the main points of this approach to correctness validation. First we introduce SSDL, closely following [13, 18].

The SOAP Service Description Language (SSDL) is a SOAP-centric contract description language for Web Services. The SOAP Service Description Language provides the base framework for a range of protocol description frameworks which at one end of the spectrum can be a simpler, SOAP-focussed, direct replacement for WSDL message exchange patterns while at the other end of the spectrum can enable formal validation and reasoning about the protocols that a Web Service supports. SOAP is the standard message transfer protocol for Web Services. However, the default description language for Web Services (WSDL) does not explicitly target SOAP but, instead, provides a generic framework for the description of network-exposed software artefacts. Another important feature of SSDL is the ability to specify multi-party protocols that are considerably more complex than the simple message exchange patterns allowed in WSDL. In SOAR we utilise the sequencing constraint manner of specifying protocols, which makes the ensuing protocol amenable to formal correctness verification.

Figure 3 and Figure 4 illustrate the use of the sequencing constraint protocol definition (the sequencing constraint schema is specified in the namespace ending with *sc*). The use of sequencing constraints results in a protocol that can be formally expressed in terms of π -calculus, thus allowing for model-checking tools to demonstrate correctness. The formal correctness proof considers the following properties: race conditions and starvation. One can also consider if an agreed-upon termination state will be reached, but we did not pursue this in this project. A race condition emerges if different participants observe different paths forward, for instance when a sender knows a message has been

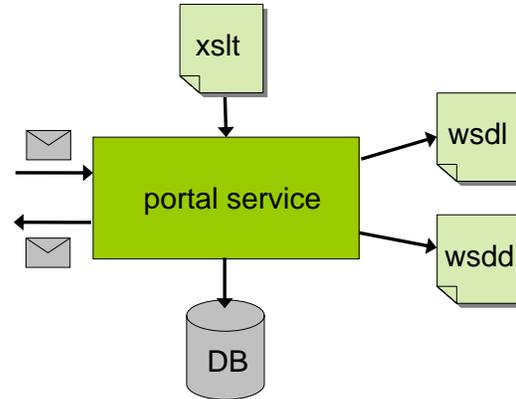


Figure 6. Service instantiation process.

sent out, while the receiver assumes no message has been sent out since it has not arrived yet. In this case, sender and receiver might take different next steps in the protocol. Starvation occurs when contracts are incompatible because certain message assumed by a receiver are not part of the protocol of the assumed sender.

We used SSDL to validate the lack of race conditions in an early version of the DealMaker service protocol specification given in Figure 3. Further details are provided in Appendix B.4.

5. Implementation and Run-Time Environment

In this section we discuss two major elements of our implementation, the service run-time environment in Section 5.1 and the SSDL protocol execution engine in Section 5.2. Extended versions of both sections can be found in Appendix B.

5.1. Service Run-Time Environment

We subsequently discuss instantiation, deployment and invocation of services.

5.1.1. Service Instantiation The concept of a portal in our architecture facilitates participants in the real-estate industry to create service instance representing them and their constituent properties. The functionality behind each of the service instances is analogous, and the sole distinguishing factor in each is the data “contained” within in. To provide a replicated service implementation for each service instance would be inefficient. We pursued a more elegant and efficient solution to this issue by providing a specialised interface to a generic service, enabling reuse of the service implementation, yet retaining the notion of distinct service instances.

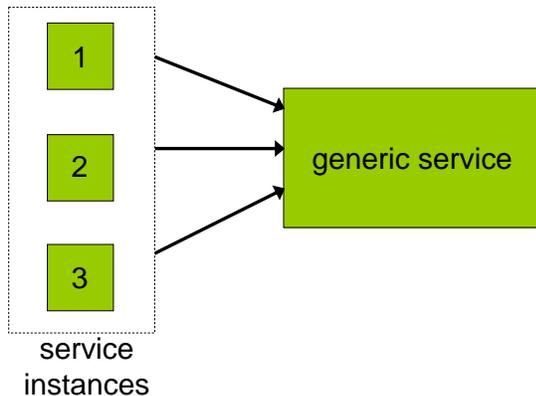


Figure 7. Service deployment process.

The production of the specialised interface, and thus service instantiation is performed by an operation at the portal service. This operation receives the instance-specific data in XML format, within a SOAP envelope, from the invoking party, be it another service or a front-end to the portal service. We use XSLT [19] to process the data received since it offers a highly effective means of focused data extraction and template incorporation. In the production of the specialised interface we wish to create for each service instance, we simply plug the instance-specific data into spaces left within a WSDL template. Within the WSDL template, the transformation simply customizes the name of the service, and endpoint at which this service was deployed.

Buyers and sellers (and other participants) can state, when registering at the appropriate portal, the service representing their lawyer, surveyor etc, for use in the deal-making process. This statement is made in the form of the URL to the given service WSDL, resulting in a list of WSDL URLs behind each buyer and seller service instance. In collecting a number of services together and making them available through a single interface, we have created a very straightforward form of service composition. Figure 6 depicts the various aspects of service instantiation.

5.1.2. Service Deployment The final output of the above instantiation process is the WSDD document. It is within this document that we compose our service instance, stating the generated WSDL as the service interface and the generic service as the implementation. Figure 7 shows how each of the created service instances is linked to the generic service implementation. With these details we have a complete description of the service instance, and therefore this service may be deployed. A tool within Axis is then used to deploy the service and enable its invocation by relevant parties. Figure 8 shows an example deployment file for the buyer and seller services.

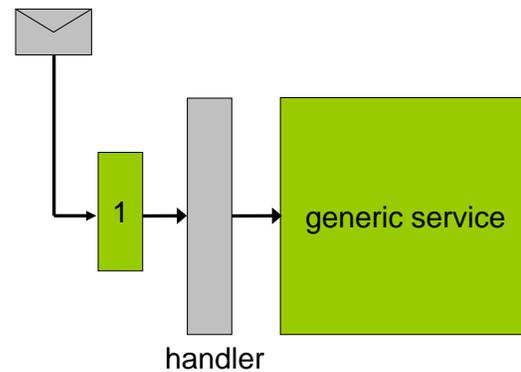


Figure 9. Service invocation process.

5.1.3. Service Invocation Deployment in above described way requires that the appropriate context be forwarded to the generic service, to enable it to distinguish invocations for different service instances. That is, given invocation of service instance A, we convey to the generic service implementation I, that context should relate to A. Therefore, all messages sent to the endpoint of a given service instance, must first pass through a handler, before being forwarded on to the generic service implementation (see Figure 9). The handler, on receipt of a message directed at a service instance endpoint inspects the message destination, that is, the endpoint of the service instance. With the use of a unique identifier for each service instance (incorporated into the instance endpoint URL), the context for a message can be derived from the message destination. This context is then added into the message body, by the handler, providing the necessary context to the generic service implementation. With this context in place, the message is safely forwarded on to the service implementation for processing. This approach shows how context for service invocation can be made implicit from the service instance endpoint rather than being included explicitly within the message. This enables the creation of replicated service instances, linked to the same service implementation, which behave as if a stand alone service with specific implementations.

5.2. SSDL Protocol Execution Engine

SSDL fully describes the state space of a composed service as well as the sequence of service interactions (message exchanges) required to reach each state. We can thus view a composed service whose description is given in SSDL as a state machine, with message exchanges providing the transitions between states, and states implicitly defined as points between these exchanges. Starting from this premise,

```

1 <deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://xml.apache.org/axis/wsdd/
  providers/java">
2   <service name="Package_n" provider="java:RPC" style="rpc" use="encoded">
3     <parameter name="className" value="scc2006.packages.PackageService"/>
4     <wsdlFile>wsdl/Package_n.wsdl</wsdlFile>
5     <parameter name="allowedMethods" value="*" />
6     <requestFlow>
7       <handler type="java:scc2006.packages.PackageHandler" />
8     </requestFlow>
9   </service>
10 </deployment>

```

Figure 8. Sample service instance deployment file.

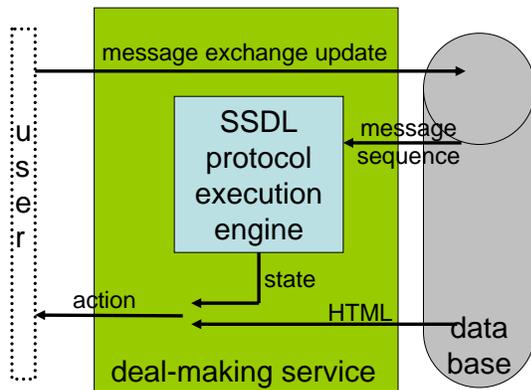


Figure 10. The SSDL Protocol Execution Engine within the DealMaker service.

we developed an SSDL Protocol Execution Engine that directly executes the state machine defined by the SSDL description.

SSDL documents describe the state space in the form of a tree whose leaf nodes are `<msgref>` elements. These specify that the type of message referenced in their `ref` attribute be sent or received. Other elements (sequence, parallel, branch, loop) define the order in which message exchanges in their subtrees need to occur. The protocol engine must correctly implement the semantic of the different elements. How this is done is discussed in detail in Appendix B.2.

The general architecture of the engine is shown in Figure 10. Based on the state reported by the SSDL Process Execution Engine, the DealMaker invokes actions tied to each state. In addition, it offers facilities to keep the internal machine state consistent with the deal's real-world status. If a user request so, based on the machine state, the DealMaker service retrieves an explanatory, pre-generated HTML page from the database and delivers it to the user.

5.2.1. State-keeping in a stateless environment Web Services that use Axis RPC wrappers are inherently stateless. Every service invocation starts with a freshly-loaded

executable. One way to keep state is by storing the input sequence that was encountered previous to reaching the current state. To restore state, the engine then steps through this sequence, ignoring actions tied to the states it traverses.

In regard to reaching the current state after startup, this method is clearly less efficient than explicit state-keeping, because all steps of the machine have to be executed again before the actual action invoked can be taken. However, we used it because (a) it is more flexible, and (b) helps to implement fault-tolerant applications (see Appendix B.2). Higher flexibility results from the fact that the state machine description (the SSDL document) can be modified between service invocations, without necessarily invalidating any partially-completed processes. This is of particular importance with long-term processes such as that implemented by the DealMaker, where one process instance may be running for several months before all steps have been completed.

6. Conclusion

This paper reports on two and a half months of team work on service-oriented computing, which included conceiving the idea of the DealMaker service, researching the real-estate business domain, designing the SOAR architecture including extensive security and trust solutions, implementing the DealMaker service and the supporting SSDL protocol execution engine, and applying model checking to a version of our protocol. The work combines state-of-the-art fundamental computer science approaches with practical implementation and with the business and standardisation side of such work. It leverages deep skills of the team members in security, protocol specification, formal methods and service-oriented software engineering, as well as the business experience of the senior team members. We believe that the current work provides a useful exploration in applying service-oriented computing technologies in the real-estate industry, with some exciting ideas and challenges we hope to continue working on in the future.

Acknowledgement

Andrew Fletcher started the contest work with us, but unfortunate circumstances left him no choice but to pull out. We have very much appreciated his enthusiastic interest in the project as well as the input he was able to give in the early design stages.

(Appendices that complete the report start on next page.)

References

- [1] Apache. Apache XML Security. <http://xml.apache.org/security>.
- [2] Apache. WSS4J XML Encryption and XMLDigitalSignature. <http://ws.apache.org/wss4j>.
- [3] Apache. *Axis Architecture Guide v1.2*. The Apache Software Foundation, 2005.
- [4] E. de Mello, S. Parastatidis, P. Reinecke, C. Smith, A. van Moorsel, and J. Webber. Demo of SOAR: Close the Deal. <http://vs-soc.ncl.ac.uk:8180/demo/index.html>, 2006.
- [5] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.
- [6] A. O. Freier, P. Karlton, and P. C. Kocher. *The SSL protocol - v.3*. Internet Draft, Março 1996.
- [7] home.co.uk. Home Buying Guide: Introduction. <http://www.home.co.uk/guides/buying>.
- [8] R. Housley, W. Polk, W. Ford, and D. Solo. *Internet X. 509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. IETF RFC 3280, Abril 2002.
- [9] MISMO. Mortgage Industry Standards Maintenance Organization. <http://www.mismo.org>.
- [10] OASIS. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v1.1*. Organization for the Advancement of Structured Information Standards (OASIS), Setembro 2003.
- [11] OpenSAML. <http://www.opensaml.org>.
- [12] OSCRE. Open Standards Consortium for Real Estate. <http://www.oscre.org>.
- [13] S. Parastatidis, S. Woodman, J. Webber, D. Kuo, and P. Greenfield. Asynchronous Messaging between Web Services Using SSDL. *IEEE Internet Computing*, 10(1):26–39, Jan/Feb 2006.
- [14] PISCES. Interoperability: The Movie. <http://.pisc.es.co.uk> and <http://www.oscres.org>.
- [15] PISCES. Property Information System Common Exchange Standard. <http://www.pisc.es.co.uk>.
- [16] Real Estate Web Sites. <http://www.imoscout.de>, <http://www.immonet.de>, <http://www.immowelt.de>.
- [17] RETS. Real Estate Transaction Standard. <http://www.rets.org>.
- [18] ssdl.org. SSDL—The SOAP Service Description Language. <http://www.ssdl.org>, 2005.
- [19] W3C. XSL Transformations (XSLT). <http://www.w3.org/TR/xslt>, 1999.

Appendices

The appendices to the above paper complete the overall contest report. In these appendices we provide more details about the following topics:

- Appendix A: the real estate problem domain and value proposition behind our SOAR and SOAR services
- Appendix B: report on implementation experiences and lessons learned, including the service run time in Appendix B.1, the SSDL execution engine in Appendix B.2, the security solution implementation in Appendix B.3, and the formal validation of SSDL in Appendix B.4
- Appendix C: explains the idea behind the demo web site in [4]
- Appendix D: a result analysis and reflection on the project

A. Problem Domain and Value Proposition

We discuss in Appendix A.1 the business case behind SOAR and individual services such as the DealMaker services, and we review in Appendix A.2 the state-of-the-art of computing technologies used in the real-estate industry.

A.1. Value Proposition

In this section we discuss the business case behind SOAR as well as individual services. It will turn out that the dynamic nature of the real-estate industry may make introduction of SOAR technologies through a winner-take-all portal relatively difficult. As a consequence, we argue that there is a role for standardisation bodies within the real-estate industry to expand their work into defining industry-wide service-oriented message interfaces.

The real-estate industry is a particularly diverse and dynamic industry. As an industry, it is not only concerned with sales and purchases of individual properties, but also of industrial properties and in response to requests for proposals in larger real-estate deals. Many parties are involved, individuals as well as businesses, with buyers and sellers continuously changing, and with a high amount of novices participating. Many steps in a real-estate transaction are traditionally human-intensive, such as viewing, decision-making, negotiating, etc. There is a continuous concern about trust and security: trust in other parties, security concerns about identity and privacy. Moreover, there is a concern of trust

in automation as well as trepidation of new buyers and sellers to step into the unknown world of real-estate. Regional knowledge is important to be a successful real-estate agent, and laws and regulations are different in every country or state, and are subject to change from time to time. Because of this diversity and dynamism in the real-estate industry, it is perhaps not surprising that automation and Internet-based cooperation are relatively slow to emerge. Compared with supply-chains or resource planning, the domain is much less straightforward to automate. However, it is exactly for these reasons that service-oriented computing solutions are required to answer the domain challenges of the real-estate business.

We believe that service-oriented computing has the potential to disrupt the real-estate industry by enabling new business practices that alter the role of current players. As an example, the role of a real-estate agent might change because of match-making and information-sharing capabilities of Internet technologies. This is already true with plain web sites, but becomes even more apparent if service-oriented solutions arise. Instead of considering this disruption a threat, we look at this as an opportunity. By adding service-oriented capabilities, new businesses can be conceived that increase the effectiveness and abilities of an existing real-estate agent. For instance, agents might rapidly create services for specific geographic areas—these service include, but are not limited to, the brochure service and listing services mentioned in the introduction.

Potential Real-Estate Services in SOAR We illustrate the potential of SOAR by describing some example applications, one of which we implemented (the ‘DealMaker service’). As a first example, assume the service to be hosted on the Internet, and assume that there is a SOAR portal that hosts information from sellers, including information about their properties, and from real-estate agents (including information about their company). Using the message interfaces of the services, one can then relatively easily create new services.

For instance, one can create a ‘brochure service’, which at the request of a buyer or agent selects a set of houses, prints them out including personalised logo and other information from a real-estate agent, and mails them to selected customers of the real-estate agent using e-mail as well as regular mail. Clearly, if such a service would have to be built using information from web sites without agreed-upon messaging interfaces, it is very difficult to build at best. There is potential for many other information-centric services that utilise the message interfaces of the portals, such as listings dependent on geographic areas, generation of targeted advertisements, etc.

Things get even more interesting when the messages not only access information, but initiate actions, such as drafting a contract or setting up a house viewing. This is illus-

trated by the DealMaker service in Figure 1. It provides the same service instance creation possibilities as the other services, but when executing, the service utilises other services to complete its process of closing a deal. Messages may for instance start the process of drafting a contract. All together, the DealMaker service does as many automatable process steps as possible to close a real-estate deal: arranging viewing dates, contacting surveyors, exchanging mortgage information, providing information for the contract, etc. The DealMaker service is described at length in Section 2.1.

Business Case for SOAR. With respect to the SOAR service architecture we introduce in this paper, there are different perspectives that one can take in judging the business validity of the approach. First, one can consider SOAR in business-to-business and even Intranet setting. In that case, one does not need to consider the difficult to control dynamics that individual customers introduce to SOAR. This greatly simplifies the bootstrapping challenge of SOAR (see below in this section for a detailed discussion of the bootstrapping issue). After all, large companies or collaborating companies can simply choose to use SOAR, because of increased efficiency and flexibility over web sites, XML interchange formats and also object-oriented methods. The trade-offs in choosing a particular architecture are similar to any other industry: ease of use, legacy issues, etc. To truly advance the state of the art, standardisation of message interfaces is necessary, either in de facto manner or through standardisation bodies such as OSCRE and PISCES [12, 15].

Although there certainly is ample reason to introduce SOAR-style architectures in business-to-business or Intranet settings, we designed SOAR with the open Internet in mind. Customers can be private individuals or businesses, thus leading to business-to-consumer as well as business-to-business scenarios. Newly provided services would be expected to aim at attracting private customer as well as small and medium businesses like lawyers, real-estate agents, surveyors, etc. All these participants will be represented by a service, which they personalise themselves through a web site. In this set-up there are two major business issues: how can SOAR be bootstrapped, and how can one make money out of starting individual added-value services such as the DealMaker service? We discuss both.

Bootstrapping SOAR. The Achilles heal of SOAR is gaining initial sustainable acceptance of the portals and the service interface definitions. We refer to this as the bootstrapping problem. The dilemma is that to start a successful portal one needs customer, and to attract customers one needs a successful portal. One obvious solution to this is to advocate the emergence of a dominating, winner-take-all, service portal—this could be a portal for each type of participant (lawyer, seller, buyer, etc.), or a portal that covers all

participants. Once the portal emerges, it can introduce de facto standards for interoperability along the lines of SOAR. This model mimics e-Bay, amazon and ‘vertical portals’ such as in the car or high-tech industry. At first sight, one might think it is a matter of time until a major real-estate portal will assume such a winner-take-all position. However, we doubt if a winner-take-all solution is likely to materialise in the real-estate industry. The real-estate industry is much more diverse than the publishing or music industry, and local cultural as well as legal aspects are different in every geographic zone. It then becomes very challenging to create a portal that is attractive for a large customer base.

If a dominant portal does not emerge, an important role arise for the standardisation bodies that represent the real-estate industry [12, 15]. Without a dominant players, de factor standardisation of interfaces is not likely to happen, and a standardisation organisation can fill the void. The organisation can also venture in creating initial portals, this resolving the SOAR bootstrapping issue. Commercial, local portals can then be expected to emerge, and creation of new value-added services would start and significantly improve the position of the real-estate industry.

Business Case for the DealMaker Service The DealMaker service is just one example of a service that can be created within the SOAR landscape. The business case behind these added services can be manifold: it can be based on charging for transactions, it can use subscription-based charging of its customers, or it could target advertisements. And, of course, a combination of these is possible. For added-value services to emerge, portals must allow messages to be sent to their hosted services, and the interfaces must be stable, preferably standardised. Here again one sees the value of standardisation, either de jure through a standardisation body, or de facto through winner-take-all portals.

A.2. Real Estate and Internet: State of the Art

As many other industries, the real estate industry is pursuing standardisation efforts to facilitate interoperability among various participants. Arguably, the real-estate industry is somewhat behind various other industries in such areas as high tech supply chains, automobile portals, etc. (the real-estate industry itself admits so in [14]). It is not unlikely that the challenges we recognised in terms of making interoperability work have something to do with the reason standards are relatively slow to take off: many different player, many different role, many different regulations, different in every geographical region. However, some serious interoperability efforts have been started, of which we mention a few.

Other obvious utilisation of the Internet is through web sites that provide information about real estate properties, allows people to advertise their property, etc. Such web sites are emerging in many countries (for some example in Germany, see [16]), states and cities, again demonstrating the dependence on geographics. These web site are a good start, but tailor to mass markets and do not provide additional interaction capabilities for software packages.

Directly related to the real-estate industry are the not-for-profit organisation Property Information System Common Exchange Standard (PISCES [15]) in Europe and the Open Standards Consortium for Real Estate (OSCRE [12]) in the United States. These organisations are publishing XML standards for interoperability between real-estate agents and possibly other parties. The agreed-upon XML formats for describing real estate related information can directly be plugged into our SOAR architecture to hook our work into existing developments. However, both PISCES and OSCRE are less concerned with the software architecture needed to deliver on some of the automation promises mentioned in the vision the expose to the world [14]. Discussions are underway what type of transport to use (e.g., ebXML or other web service technologies), but that stops far short from a service-oriented solution.

Many other standardisation efforts are of interest, to name a few, the Mortgage Industry Standards Maintenance Organization (MISMO [9]) for the mortgage industry or the Real Estate Transaction Standard (RETS [17]). The latter is centred around a server platform solution, but in general these organisations propose standardised formatting of data in XML, so that software programs can easily interoperate. We have argued in this paper that new opportunities arise if one goes beyond format and looks towards flexible ways of enriching the interaction between participants. Nevertheless, a very important role could be in store for the standardisation bodies to push the XML standardisation of message-based interactions. The creation of XML standards for the message interchanges by standards bodies would be an alternative to the emergence of large portals that create their own message interchange definitions which will turn into de facto standards. Because of the diversity for the industry, large commercial players may be slow to emerge, in which case there is a role for standardisation bodies to push SOAR like solutions for the real-estate industry.

B. Implementation Experiences and Lessons Learned

We implemented our solution using Java technologies, a choice motivated by the familiarity with the technologies and the existence of diverse tool support. The demo runs

on a web server hosted at The University of Newcastle, relying on Axis and on a MySQL database. Axis and related software runs in VMWare virtual server on Linux (offering a Linux guest OS for our use), connected to the database hosted on another machine meant for student projects. Note that our service implementation is not distributed over many hosts but are web service all on the same host (identified by URLs). The choice of individual development environments were left to each developer in the team. We made limited use of UML techniques to describe and share designs, without any particular tool support in that area. The web site was implemented by relatively standard means: dynamic web pages, using 'AJAX'-style interaction through Javascript and passing of XML documents.

B.1. Service Run-Time Environment

This subsection is an extended version of Section 5.1.

The SOAR implementation necessitates a run-time environment capable of supporting the characteristics synonymous with the service-oriented paradigm. The architecture used to support service-orientation characteristics is web services, and principally the SOAP-based implementation of this architecture. Numerous tools supporting the development of web services have been born out of this position at the forefront of service-orientation, and the SOAR implementation takes advantage of one such tool, namely Apache Axis [AXIS]. Axis provides a sound basis for web service development with, most notably, the provision of a SOAP processing and transport framework and flexible service deployment options. This utilisation of Axis enabled a certain degree of abstraction to be achieved in the development process, with focus shifting, as far as possible, to higher level and more conceptual notions.

Standard techniques for web service instantiation, deployment and invocation are ubiquitous, as the adoption of the service-oriented paradigm and web services architecture gains ever-increasing momentum. Consequently, this section places focus on the innovative, non-standard techniques used in our implementation approach in relation to service instantiation, deployment and invocation.

B.1.1. Service Instantiation One fundamental requisite of a web service is the decoupling of interface from implementation, facilitating opaque invocation of web services and a focus on what functionality is provided not how it is provided. The interface states the operations and message formats supported by the service, and the endpoint at which this service resides. The service consumer is abstracted away from service implementation details, and concerns himself with only matters prior to message dispatch to the endpoint. Such abstraction provides a high degree of flexibility to the service provider, allowing service implementation to be arbitrarily complex whilst maintaining a

consistent and abstract interface. In the SOAR implementation we capitalise on the flexibility offered by this decoupling, and exploit the power of this technique for service instance creation at run-time.

The concept of a portal in our architecture facilitates participants in the real-estate industry to create service instance representing them and their constituent properties. One can view this creation as a factory-style process, a participant registers at the portal, and portal creates a service instance for them representing their particular participant type, for instance the buyer portal would create a service instance for a buyer. The functionality behind each of these service instances is analogous, and the sole distinguishing factor in each is the data contained within in. To provide a replicated service implementation for each service instance would not only be highly inefficient, it would also be contradictory to the core principles of service-orientation; replicating rather than reusing functionality. A more elegant and efficient solution to this issue would be to provide a specialised interface to a generic service, enabling reuse of the service implementation, yet retaining the notion of distinct service instances.

The production of the specialised interface, and thus service instantiation is performed by an operation at the portal service. This operation receives the instance-specific data in XML format, within a SOAP envelope, from the invoking party, be it another service or a front-end to the portal service. Contrary to all other service operations within the SOAR implementation, this operation utilised document-style, literally encoded SOAP messages. The justification for this stems from our wish to use XSLT [19] to effectively and concisely process the data received. RPC encoded messages would not be suitable for this purpose, as their component data is isolated on receipt and made accessible as atomic data items.

XSLT offers a highly effective means of focused data extraction and template incorporation. This makes it highly effective in the production of the specialised interface we wish to create for each service instance simply plugging the instance-specific data into spaces left within a WSDL template. The functionally analogous nature of the service instances meant that a set WSDL template contained all the pre-defined operations and message formats, and the transformation simply customized the name of the service, and endpoint at which this service was deployed. The instance-specific data “behind” the service instance must be stored in a database to enable its retrieval and amendment by subsequent instance invocations. Each service instance is assigned a unique identifier and the data is linked to this identifier within the database. We again perform the task of data extraction with the aid of XSLT, which plugs the extracted data into a pre-formed SQL statement and updates the database accordingly. Use of templates in this way offers a high

degree of flexibility to change, as new templates can be plugged in as requirements evolve.

The final output of the instantiation process is the WSDD document, generated using precisely the same method as the WSDL document, through use of a template in XSLT. Axis offers flexible deployment through customizable options with this WSDD document, enabling the definition of numerous service-specific details including the interface and implementation corresponding to this service. It is within this document that we compose our service instance, stating the generated WSDL as the service interface and the generic service as the implementation. Further explanation of the deployment process is left to the next section.

Worthy of further discussion is the instantiation of buyer and seller service instances. Buyers and sellers can state, when registering at the appropriate portal, the service representing their lawyer, surveyor etc, for use in the deal-making process. This statement is made in the form of the URL to the given service WSDL, resulting in a list of WSDL URLs behind each buyer and seller service instance. These addresses are stored within the database along with any other instance-specific data and can be extracted for use in the deal-making process. In collecting a number of services together and making them available through a single interface, we have created a very straightforward form of service composition.

One may, of course, provide a specialised, custom implementation for a given service instance, as would most likely be the case for mortgage lenders, lawyers etc. For the case of buyers and sellers though, it is unlikely that the resources and knowledge available would enable them to configure a specific web service for themselves. Our approach, therefore, strives to illustrate the elegance and efficiency with which functionally analogous service instances can be created by services at run time, using pluggable templates. This holds many opportunities both within and outside of the real-estate industry.

B.1.2. Service Deployment As discussed in the previous section, one output of the instantiation process is a WSDD document, enabling the custom deployment of created service instances. The document generated is used by Axis to correctly deploy the service, and to route service invocations to the appropriate service implementation.

Figure 7 shows how each of the created service instances is linked to the generic service implementation. We establish this configuration in the WSDD document, stating the generic service implementation as the implementation of the service instances, and the generated WSDL as the interface for this service. With these details we have a complete description of the service instance, and therefore this service may be deployed. A tool within Axis is then used to deploy the service and enable its invocation by relevant parties. Figure 8 shows an example deployment file for the

buyer and seller services. Deployment in this way requires that the appropriate context be forwarded to the generic service, to enable it to distinguish invocations for different service instances. We discuss this notion of context with regard to service invocations in the next section.

B.1.3. Service Invocation This invocation model of web services can be extended with the notion of handlers. Handlers enable web services to define a functional intermediary in the invocation process to intercept all incoming and/or outgoing messages, execute some functionality, and on completion forward the message on. This functionality is commonly used to enforce security or trust procedures before the invocation of a web service, but within the service instance creation process we use these handlers to convey context.

Our use of a generic service implementation for multiple service instances requires that context be conveyed. That is, given invocation of service instance A, we must convey to the generic service implementation I, that context should relate to A. Of course, this could simply be included in the SOAP communication to the service instance, but this is contradictory to the idea of generating a specific service instance. In such a case, we could simply have one generic service interface and implementation, and pass instance context as a parameter to this service, in the form of the instance identifier. The SOAR implementation endeavoured to create a more elegant and useful approach to this context communication, and found such an approach in the use of handlers.

Our context for each instance was the instance identifier, and it was this identifier we required to be conveyed to the generic implementation. Messages arriving at the service instance endpoint had no containing context, that is, the context was implicit from the endpoint at which the message was directed. For instance if we dispatch a message to the endpoint of service instance A, we do not include within the body of that message any reference to service instance A. If such a message was then simply forwarded on, without amendment, to the generic implementation, we would be unable to derive the message context, that is, the instance to which this message relates.

To deal with this notion of context we introduced a handler to the invocation process. All messages sent to the endpoint of a given service instance, must first pass through this handler, before being forwarded on to the generic service implementation. The handler itself is generic, and the same handler is utilised by all service instances, and in essence this handler can be seen as part of the generic implementation. The handler, on receipt of a message directed at a service instance endpoint inspects the message destination, that is, the endpoint of the service instance. With the use of a unique identifier for each service instance (incorporated into the instance endpoint URL), the context for a message

can be derived from the message destination. This context is then added into the message body, by the handler, providing the necessary context to the generic service implementation. With this context in place, the message is safely forwarded on to the service implementation for processing. Such an approach has shown how context for service invocation can be made implicit from the service instance endpoint rather than being included explicitly within the message. This enables the creation of replicated service instances, linked to the same service implementation, which behave as if a stand alone service with specific implementations.

B.2. SSDL Protocol Execution Engine

This section is an extended version of Section 5.2.

There already exist various tools to support the use of SSDL in the development of new web services. These tools provide facilities for correctness-checking SSDL descriptions and for the automated creation of .Net stubs from SSDL. As with most similar approaches throughout the field of computer science, however, a gap opens between formally checked descriptions and their actual implementation, because the components involved must still be developed manually. Hence, mistakes during the implementation could re-introduce protocol errors easily found (and fixed) in the formal description; and consequently deployed services are still prone to these kinds of errors.

On the other hand, SSDL fully describes the state space of a composed service as well as the sequence of service interactions (message exchanges) required to reach each state. We can thus view a composed service whose description is given in SSDL as a state machine, with message exchanges providing the transitions between states, and states implicitly defined as points between these exchanges.

Starting from this premise, we developed an SSDL Protocol Execution Engine that bridges the aforementioned gap between description and implementation by directly executing the state machine defined by the formal description. For each message exchange observed, the machine's state is advanced to the state specified in the description. Then, an action tied to this state can be invoked. This action leads to another message exchange, which in turn advances the state machine to the next state.

B.2.1. Implementation of SSDL Elements SSDL documents describe the state space in the form of a tree whose leaf nodes are `<msgref>` elements. These specify that the type of message referenced in their `ref` attribute be sent or received. Other elements define the order in which message exchanges in their subtrees need to occur. In respect to their influence on the state machine's behaviour, these fall into four classes:

Sequential Execution All child nodes must be executed sequentially, i.e. in the order they are given in by the SSDL document. A node with sequential execution is considered completed when all message exchanges required by its children have taken place. This class comprises the `<sc>`, `<protocol>` and `<sequence>` elements.

Parallel Execution The order in which child nodes are executed does not matter, but as with sequential execution all message exchanges must be completed before a parallel execution node is complete. This class is made up of the `<parallel>` element.

Branches Exactly one of the child elements must be completed. If the special `<nothing>` element is present, a branch may be skipped. I.e., execution of the `<nothing>` element implies that none of the other child nodes must be visited, hence none of the messages specified in their `<msgref>` descendants must be observed before the branch can be completed. The `<choice>` element is the only member of this class.

Loops All child nodes can be executed multiple times and in parallel, i.e. one loop need not be finished before the next starts. Loops are specified by the use of the `<multiple>` element. We do not support loops in our current implementation.

Our SSDL Protocol Execution Engine recursively visits and marks completed nodes according to the order of message exchanges observed and required by the SSDL description.

B.2.2. State-keeping in a stateless environment We developed the DealMaker service within an Axis environment. Web Services that use Axis RPC wrappers are inherently state-less: Every service invocation starts with a freshly-loaded executable. Services that need to keep their processing state between invocations have to save and restore the information necessary to do so to/from an external storage system (e.g. a database).

We distinguish two ways of keeping state. First, the state itself can be saved explicitly. With our engine implementation, this corresponds to saving the current state space tree, whose configuration of completed and uncompleted nodes represents the state machine's state. In an implementation, this provides a reasonably efficient method to reach the current state before continuing work.

Second, with a state machine that is driven solely by external input, state can also be kept by storing the input sequence that was encountered previous to reaching the current state. To restore state, the engine then steps through this sequence, ignoring actions tied to the states it traverses. In regard to reaching the current state after startup, this method is clearly less efficient than explicit state-keeping, because

all steps of the machine have to be executed again before the actual action invoked can be taken. However, we favour it because (a) it is more flexible, and (b) helps to implement fault-tolerant applications. Higher flexibility results from the fact that the state machine description (the SSDL document) can be modified between service invocations, without necessarily invalidating any partially-completed processes. This is of particular importance with long-term processes such as that implemented by the DealMaker, where one process instance may be running for several months before all steps have been completed.

Furthermore, storing and re-reading the input sequence offers an obvious starting point for the application of fault-tolerance (FT) measures. N-Version Programming (NVP) as a means to improve the reliability of the SSDL Protocol Execution Engine itself illustrates this best. In short, NVP entails the use of several different implementations of the same component to eliminate errors introduced during the programming process. With input sequences stored and available in the same format to each version, individual implementers can concentrate on improving the core engine, and avoid inter-version dependencies in the state-keeping code.

B.2.3. Setting-specific implementation details We previously described the SSDL Process Execution Engine on an abstract level, considering message sequences as its input and unspecified 'actions' as what happens in the single states. In the following, we will point out several details of the implementation as part of the DealMaker service.

The general architecture is shown in Figure 10: Based on the state reported by the SSDL Process Execution Engine, the DealMaker invokes actions tied to each state. In addition, it offers facilities to keep the internal machine state consistent with the deal's real-world status.

At the moment, our implementation only performs one type of action: The user is presented with the state of the deal and with his options to progress it. Based on the machine state, we retrieve an explanatory, pre-generated HTML page from the database and deliver it to the user. While limited in its general applicability, this choice is adequate for the human-centric interactions that dominate our business case. In the future, invocations of services that help the user complete his deal could be tied to some states; e.g. a service that negotiates between schedules might help buyers and sellers set a date for the viewing of a property.

To keep the internal machine state consistent with the real-world status of the deal, the DealMaker must be aware of any message exchange that takes place between parties within the protocol. The most straight-forward way to achieve this is to implement the DealMaker as a message broker for all messages sent during the deal. However, this does not only involve privacy issues and performance considerations that may both hamper acceptance of the service,

but also reduces service flexibility by tying parties to one central entity.

We therefore chose to simply offer an interface for the user to notify the DealMaker that they have sent (or received) a specific type of message, i.e. that a message exchange has occurred. Note that the actual message contents are not of importance here; the DealMaker needs to know only the type of message that was sent. As we keep state solely through the sequence of messages encountered, the DealMaker only has to store the kind of message it received into the database.

At the core, both ways in which the DealMaker interacts with the outside world are implemented as Web Services using SOAP. External parties can both query and update the current deal state through the DealMaker. Access to these methods is possible in the standard Web Services fashion (i.e. by sending and receiving SOAP messages). The WS interface simplifies the creation of external services that make use of the DealMaker's functionality. In fact, our web interface, which hides these technical details from the human user, is implemented as an in-browser WS client sending/receiving SOAP messages.

B.3. Security and Trust Implementation

Messages between services carry security information, like identification or rights. In SOAR we express this kind of information using SAML authentication and attribute assertions. To make sure that the application layer does not need to understand about security information, the security solution is based on Axis handlers [3], that intercept in a transparent way SOAP messages exchanged among service instances. SAML assertion can be inserted and checked without interaction with applications (or, in fact, with users). The security implementation was done using the follow open source libraries: Apache XML Security—an implementation of XML Encryption and XML Digital Signature [1]; WSS4J—a WS-Security implementation [2]; and OpenSAML—a SAML implementation [11].

B.4. Model Checking SSDL

SSDL was chosen as protocol and message exchange definition language to utilise the proving capabilities of the sequence constrains form of protocol specification. There is no automated tool support yet for model checking SSDL protocols, so we translated the specification by hand in the form of a π -calculus. In this way we were able to validate an earlier version of our protocol for correctness—the final version of our protocol should be checked again. Obviously, there is a great need for SSDL-related model-checking tool support to make it practical to check SSDL specification throughout the design phase, especially in light of the fact

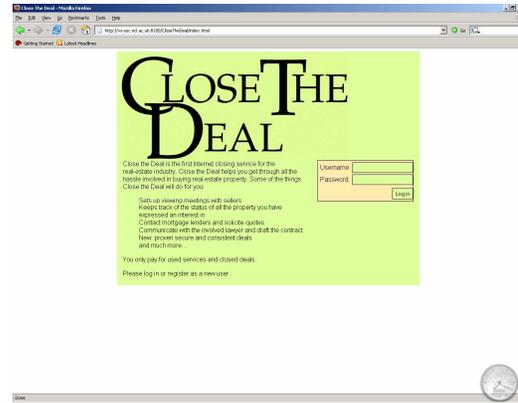


Figure 11. The demo welcome page.

that these specification may be altered at various time, for instance because of implementation decisions that alter or limit the protocol behaviour.

The need for tool support for SSDL is prevalent throughout the design phase. Existing tool support [18] creates typed representations of messages, supporting both C# and Visual basic code generation, as in addition validates the specification for consistency. However, the tools do not take the protocol framework of SSDL into consideration. Therefore, the most important novel implementation task in our system was the design and implementation of the SSDL protocol engine, which we described in detail in Section 5.2.

Because of time pressure, we were not able to validate the final version of our DealMaker service, as given in Figure 3. In SOAR interesting issues arise with respect to determining the state of multiple service instances concurrently. Each service instance itself runs a copy of the DealMaker protocol, but customers as well as the device run-time can be interested in multiple service instances at the same time. For instance, in the demo, we would like to check that at most one of the instances of each customer gets into the final phase—otherwise, the buyer would end up with multiple properties. Or, we would like to make sure real-estate properties are not sold twice, an even harder problem because it not only goes across multiple instance, but also multiple customer (i.e., buyers). Other states defined across multiple service instances could be thought of, and it would therefore be of interest to identify ways to express states of interests and execute the model checking in efficient manner across multiple instances at once. We have had to leave this for further research.

C. The Demonstration Web Site

Note that this description might be subject to change in some of the specifics.

In this section we detail the intent behind the demonstration web site (see Figure 11), and provide a manual of the actions one can execute on the web site. The focus of the web site is on demonstrating the viability of the technologies we applied—we have made no concerted effort to make a product-quality web site. It should be noted that the web site has mostly been tested for the Firefox browser—we strongly suggest one to use the Firefox browser, since we cannot provide any guarantee about correct operation for other browsers.

The purpose of the web interface is to demonstrate how a potential buyer would utilise the ‘Close The Deal’ real estate closing service to help with the process of buying real estate property. In what follows, we use ‘user’, ‘buyer’ and ‘customer’ to mean the same thing: the person who logged into the web site. The fact that we focus on the buyer has one immediate consequence: the user interface is limited to a view for the buyer, even though the service design support activities of other participants as well (the seller in particular). It should also be noted that the buyer is not aware of the fact that it uses a service-oriented computing architecture—on the contrary, this remains hidden. The only way in which a buyer would notice that the supporting implementation uses advanced technologies is through the advanced functionality the web site delivers.

Note that every time you log in to the web site, you start in the same process state!

A buyer that uses the Close The Deal web site needs to do two administrative steps (or one, in case of a returning customer), and then is guided through the process of closing real estate deals, for as many deals as the user desires. The two administrative steps are:

1. *Logging in.* Every time a user enters the web site, he/she needs to log in. This can all be implemented by standard means. We only use it to limit concurrent and ill-fated access to the web site, and support only one user, named `close` with password `thedeal`. Please use this account to view our web site.
2. *Provide buyer details.* A regular form needs to be filled out with information about the buyer. The interesting aspect of this is that one can fill in data that corresponds to a service instance in the SOAR architecture. For instance, one chooses a lawyer from a list of lawyer, each of which is represented by a service instance in our system. Note again that the user does not know it selects a service instance—the user only could realise the efficiency of the architecture when he/she realises there is no need to fill in any additional information about the lawyer (or other participants) after selecting it.

The interesting aspect for a customer comes when selecting houses in which it is interested. The user therefore

must construct a *deal*, which the service then tracks. To construct a deal, the customer browses potential offers, and selects the one it is interested in. As a consequence of this set-up, there are two ‘deal-making actions’ a customer can select:

1. *Browse and select real-estate properties.* When the customer clicks the `Browse Offers` button, the browser displays a list of properties, of which one at a time can be selected. Selection of a property is simply done by clicking on the hyperlink located with the property.
2. *Check progress of the deal.* When the customer clicks the `Check Status` button, it provides the list of deals in progress. Important in this list is the *status* of each deal: how far along is the customer in closing the deal. Each deal goes through the steps of the business process, but the user is continuously involved in providing feedback on whether and how a next step needs to be carried out. The start state of our web site provides the status `initiate`. Once you click that, you are asked if you want to start the mortgage application, for the amount given in your profile or the asked price, whichever amount is larger (which, unbeknown to the buyer, is a property of a service representing the buyer). If you agree, the Close The Deal service will request mortgage lenders if they want to provide a mortgage. In the demo, you will simply have to wait for the process to finish (which in real life could take hours or even days), so essentially the demo stops here. However, this clearly demonstrates how the buyer interacts with the Close The Deal service to help getting through the process of buying a house. Follow-up steps, which we in fact also implemented the machinery for, include arranging a viewing meeting, contacting a surveyor and drafting a contract with a lawyer, along the lines of the protocol in Figure 3.

D. Reflection

The complete scenario and work described in this report was conceived, designed and developed within an eleven weeks time span. We briefly want to reflect on the main challenges we faced during the project: conception of the real-estate scenario, and the geographic distribution of the team members. We also review the business as well as research opportunities sprouting from the reported work.

D.1. Conception of SOAR

The open-ended nature of the contest brought a challenge as well as an opportunity to the work we would

be able to do. The choice of application area (real-estate) was partly motivated by existing contacts with the PISCES real-estate standardisation organisation located in Newcastle. However, the scenarios, business case and usage models were all developed from scratch. We have tried to utilise the industrial experience of the senior team member in creating business models for SOAR and SOAR services, but were mostly motivated by the technical questions 'why services' and 'how to do services'. In terms of time, the scenario conception took 40 percent, the design 20 percent, and the implementation 40 percent, but obviously the boundaries between these phases are loosely defined. It would be of great interest to us to continue some of the technical work, and a future opportunity to work on the SOAR architecture without a lengthy phase of conceiving scenarios would give us an opportunity to show even more exciting technical results. The sound technical skills and deeply developed intuition for service-oriented computing that is present in the industry advisors within our team was leveraged heavily, and is at the heart of the work on SSDL and the associated protocol execution engine. This provides us with some unique technologies that we feel are very much worth pursuing further within the SOAR context.

D.2. Geographic Distribution of Team

Our team was assembled solely for this contest. Timing-wise, the lucky opportunity arose for the people involved to spend time on the contest (in varying amounts). We participated with the objective to use the contest as a learning experience, as well as a source of fun by working as a team toward a common goal with a competitive element. In spirit with the openness associated with service-oriented computing, our team is globally distributed: the six team members are located in five different countries, four different continents. There are two industrial advisors (Jim and Savas) with a lot of experience in service-oriented software—these people also developed SSDL. Aad has close to a decade of industry experience and brings in some domain knowledge through contacts with the real-estate standardisation organisation PISCES [15]. Emerson is a PhD student specialising in security, Chris a PhD student specialising in software for distributed decision-making, and Philipp a master student who has published on web service reliability. The latter three implemented the system.

The distributed nature of the team provided obvious communication challenges we had to learn to deal with. From our industry experiences, we were aware of the challenge in starting new projects when a team is geographically dispersed, and it took us quite some effort to find a good way of producing as a team. For the purpose of the contest, we travelled to create periods in which all core members of the team were in Newcastle to work full-time on the project.

This provided excellent results, and to our judgement periods of geographic collocation are a prerequisite for a team development project that starts from scratch. It did become clear, however, that technologically we still have a long way to go before Internet communication tools effectively support (in cheap and robust way) the kind of communication and interaction required for a high-pace concentrated team effort like ours.

D.3. What is Next?

It would be exciting to continue the work we started for this contest. There are opportunities to go after the business ideas presented in this report, and there exist opportunities to influence the standardisation bodies. From a technical perspective, the connection of human interaction and web services has been an eye opener, and is of interest to pursue further. The implementation ideas that relate to the hosting of many personalised service instances in an efficient manner need to be explored further. Possibly design patterns can eventually be derived from our solutions. Finally, we have gained considerable insight in the working of SSDL, which we would like to utilise to further improve that technology. This also includes improving the abilities to proof correctness of protocols with respect to concurrent service instances.

Survey of Service Discovery Protocols in Mobile Ad Hoc Networks¹

Technical Report 7/06

Adnan Noor Mian, Roberto Beraldi, Roberto Baldoni

Dipartimento di Informatica e Sistemistica “Antonio Ruberti”
Università degli Studi di Roma “La Sapienza”
Via Salaria, 113 – 00198
Rome, Italy

{Adnan, Roberto.Beraldi, Roberto.Baldoni}@dis.uniroma1.it

Abstract. Mobile devices are inherently scarce in resources, necessitating the need to cooperate among them for performing tasks that cannot be done alone. This cooperation is in the form of services that are offered by other devices in the network. To get benefit from the services offered by other devices, they have to be discovered. Service Discovery Protocols (SDPs) are used for this purpose. This is an important area of research in mobile and ubiquitous computing. In this paper twelve SDPs for multihop mobile ad hoc networks are analyzed with respect to their service discovery architectures, management of service information, search methods, service selection, methods for supporting mobility and service description techniques. Among these the most important aspect is the service discovery architecture as this affects other aspects of the service discovery. In this paper the service discovery architectures are categorized in four groups namely directory-based with overlay support architecture, directory-based without overlay support architecture, directory-less with overlay support architecture and directory-less without overlay support architecture. The management of service information and search methods mainly depend on the type of service discovery architecture. It is found that mobility support and service selection methods are independent of the SDP architecture. Also the description of services is found to be independent of the SDP architectures. Mostly the services are described using XML or the extensions of XML. At the end of the paper open issues and areas of further research are discussed.

Keywords: Mobile Ad hoc Networks, Overlay networks, Service Discovery and Service Discovery Protocols

¹ This work was supported by the Italian Ministry of Education, University, and Research (MIUR) under the “IS-MANET” project and by the NoE “RESIST” (European Commission 026764)

1 Introduction

A service can be any tangible or intangible thing that can be useful for someone. For example when a laborer works for building a house, he is giving his services for which he is paid. Similarly the act of teaching is a service provided by a teacher to his students. In our context of mobile ad hoc networks any facility provided by a device that can be useful for any other device is a service. A service in this context could be a software service like providing an implementation of some algorithm (for example, converting one audio file format to another) on a device so that when some device needs this service, it can contact that device and use it. A service can also be a hardware service like a printer that can be used by a mobile device to print a file. To get benefit from these services a device must be able to locate them in the network and also have the ability to invoke these services. Here comes the role of service discovery protocols.

In fixed and wired networks service discovery protocols simplify the interaction among users, devices and services [8]. Service discovery protocols allow devices to automatically discover network services thus making the task of network administration and configuration easy.

In wireless mobile ad hoc networks devices are free to move. The characteristic limitation of a mobile device is that it has to be small in size and weight. Such devices inherently have few and limited number of resources as compared to fixed devices. So it becomes important to utilize the resources and services available in other devices to accomplish the tasks that cannot be done alone. For example a mobile device without a printing support will require a printer to fulfill the printing task. Thus forming ad hoc network between mobile devices and getting benefit of resources available in a network require knowledge of services available by other devices and how to interact with these services. The service discovery protocols aim at these aspects. More specifically the service discovery protocols not only provide mechanisms for locating a particular service but also mechanisms to advertise a service, invoke a service, select a service if there are more than one services of the same type available and to describe a particular service so as to make its searching easy.

There is a lot of research work going in the field of service discovery. Basically there are three types of networks as far as the research in service discovery is concerned. First are the wired networks, second are single hop wireless networks and third are the multihop wireless mobile ad hoc networks. The service discovery protocols suggested for one type of network are not suitable for another type of network because each network is based on different assumptions, the most important being the mobility and rate of joining and leaving of devices from the network. In the first type devices do not move at all and there is no join/leave at all or the join and leaves are few and far between. In the second type the network formed is ad hoc with very restricted mobility and having low rate of join/leave. There are one or more nodes that are fixed. But in the third type of network the devices are assumed to have unrestricted mobility and these can join or leave the system at any rate. There may be no fixed node. Due to these assumptions the problem of service discovery is very challenging in the third type of networks.

In wired networks many service discovery protocols have been suggested, some of which have gained the status of industry standard. For example Jini [17] by Sun, Universal Plug and Play (UPnP) [18] by Microsoft, Salutation [19] by IBM and Service Location Protocol (SLP) [20] by IETF. Mainly the service discovery protocols for wired networks fall into one of the three categories. Some are directory-based, like Jini that has a centralized place to store information about the services. Some are directory-less like UPnP that has a peer-to-peer (P2P) architecture and the service information is stored on each device. The third category is the hybrid of the above two. For example SLP can work in both modes, that is with or without a directory depending on the situation.

In single hop ad hoc networks there are also some mature protocols available. For example Bluetooth SDP [22] and DEAPSpace [21]. Bluetooth's SDP is an industry standard. These protocols may follow P2P architecture like DEAPSpace or a client-service approach like Bluetooth SDP.

In multihop Mobile Ad hoc Networks (MANETs) a lot of research is going on but still it has not been mature enough to be used by industry. The main reason for this lack of mature research, in spite of a lot of efforts by the research community, is the challenging issues due to the unrestricted mobility of devices. A lot of work has been done in the field of routing in MANETs. One can take advantage of this work for studying the service discovery problem. But essentially the service discovery problem is different from the routing problem. In routing we know the ID of a node, which is unique and data is only sent to that particular device, whereas in service discovery there is a service, which may not be unique and its multiple copies can reside on different devices. The task is to find that service, which best fits some given criteria. A service discovery protocol (SDP) may use an underlying routing protocol to invoke and get a reply from a particular service residing on a particular device. There are some SDPs that integrate the functionality of routing and service discovery. Thus service discovery and routing although are quite related to each other but specifically have distinct characteristics. One can take advantage of the research work going on in one field for the benefit of the other.

There are some good surveys of the service discovery protocols that also include SDPs for MANETs. For example the surveys done by Cho and Lee [14], Zhu and Mutka [16] and Marin-Perianu, Hartel and Scholten [15]. These surveys survey the service discovery protocols for all of the three types of networks and none of the surveys go deep into surveying service discovery protocols for only the multihop mobile ad hoc networks. In this paper we have focused on SDPs for MANETs only. For this purpose we selected twelve SDPs that have been referenced quite often by other authors. These are protocols by Cheng et al [1], GSD [2], Allia [3], Konark [4], Service Rings [5], Lanes [6], protocol by Kozart et al [7], Splendor [8], protocol by Varshavsky et al [9], protocol by Tyan et al [10], Field Theoretic Approach [11] and DSD [12]. We have selected six major components or aspects of any SDP and then analyzed the selected SDP with respect to these aspects. These aspects are service discovery architectures, management of service information, search methods, service selection methodologies, mobility support mechanisms and service description. From our point of view the most important aspect of any service discovery protocol is its architecture as other aspects may also depend on it.

The paper is organized as follows. In section 2 service discovery architectures are discussed. After discussing the existing two types of categorizations, another categorization is proposed that is a better representation of service discovery architectures of MANETs. It is also shown how different MANET SDPs fit in this categorization. In section 3 and section 4 the management of service information and search methods respectively are discussed. Service selection methodologies and the metrics used for algorithm based service selection are discussed in section 5. Section 6 is devoted for describing methods for dealing with mobility in MANETs. First the conditions in which a system of mobile nodes need mobility support is explained and then three methods for supporting mobility in existing SDPs are presented. Section 7 describes three different trends in service description techniques. Finally section 8 presents the conclusions and areas of future work.

2 Service discovery architectures

Broadly speaking architecture specifies the layout of any structure and how major components of that structure are connected with each other. The SDP architecture mainly depends on having or not having a directory. A directory is an entity that stores information about services available in the network. It helps in the service discovery process [14]. With respect to the directory the architectures for any service discovery protocol could be directory-based, directory-less and hybrid of the two. C.K. Toh [13] has already categorized three possible service discovery architectures in MANETs. These architectures are service coordinator based, distributed query-based and hybrid of these two.

In service coordinator based architecture, which is similar to directory-based architecture or centralized directory as called by Cho and Lee [14], certain nodes in the MANET are chosen to be Service Coordinators (SCs), a role quite similar to the Directory Agent (DA) in SLP [20] or the lookup service in Jini [17]. SCs announce their presences to the network periodically by flooding SC announcement messages. The flooding is limited to a certain number of hops, determined by the SC announcement scope parameter. Directory-based protocols include Service rings [5], Splendor [8], protocols by Kozart and Tassiulas [7] and Tyan and Mahmoud. [10].

In distributed query-based architecture, which is same as the decentralized directory or directory-less as Cho and Lee [14] has called it; there are no such Service Coordinator nodes. Instead, a client floods the service discovery request throughout its surroundings in the network. The flooding is limited by the flooding scope parameter. Directory-less approach is used far more than directory-based approach in MANETs. Examples of this approach are GSD [2], Allia [3], Konark [4], Field theoretic approach [11], protocols by Cheng and Marsic [1], Varshavsky, Reid et. al. [9] and DSD [12] by Chakraborty, Joshi et. al.

The hybrid architecture combines the above two architectures. Service providers register their available services to one or more available SCs. These SCs are also ready to respond to flooded service requests. When a client unicasts a service request to its affiliated SC according with the service coordinator based architecture,

the SC responds with a positive or negative service reply. However, if there is no SC in the client's surroundings or if the affiliated SC returned a negative service reply, the client will simply fall back to the Distributed query based or directory-less architecture. Hybrid architecture is mostly found in wired networks and there is no real example of such type of SDP architecture in MANETs.

The SDP architectures for MANETs can also be categorized by having an overlay network support or not having an overlay network support. An overlay network can be explained as follows. If a node, which is a part of an ad hoc network, knows the address of another node of the same network and can communicate with it, then we say that there is an overlay link between the two nodes [5]. An overlay link does not necessarily mean that the two nodes have a direct physical or wireless connection. Two nodes can form an overlay link even if they can communicate through many intermediate nodes. An overlay network is a collection of such overlay links and the nodes they connect. The overlay can be a structured overlay network when there is some organization between the nodes forming the network or it can be unstructured in which there is no organization and the nodes are connected randomly. In both cases, that is structured or unstructured overlay networks there is a bootstrapping phase in which the nodes form the overlay network. Also there are special algorithms for joining and leaving of nodes from the overlay network. Note that there is a difference between unstructured overlay network and a network that does not have an overlay at all. In an unstructured overlay network firstly, a node forms overlay links to nodes, that is, it has the addresses of nodes beyond the neighbor nodes and can communicate with those nodes and secondly, these overlay links are randomly connected with the node. In case of not having an overlay network, a node has only the addresses of its neighbors and does not have addresses of nodes beyond its neighbors. Thus a node can only communicate directly to its neighbor nodes. From neighbors we mean all nodes that are in the radio range of a node.

Also note that whenever there is an overlay network support, it will always be a structured overlay and not an unstructured overlay. The reason for this is that the structured overlay network has the advantage of controlled multicast of service query or advertisement message. This controlled multicast restricts and greatly reduces the network traffic. Thus, although we pay for forming and maintaining the structure but also get an advantage of reduced network traffic. In case of having unstructured overlay networks in MANETs there is no such advantage of reduced network traffic but the cost of forming and maintaining the overlay is always there, although it may be less than the previous case. We thus have to pay but without getting any advantage. Therefore it does not make any sense to use unstructured overlay networks in MANETs. Examples of protocols forming overlay networks (which are structured) are Allia [3], Service rings [5], Lanes [6], protocols by Kozart and Tassiulas [7] and Tyan and Mahmoud [10].

The SDPs that do not form overlay, do not have a bootstrapping phase or special algorithms to maintain the overlay structure. Thus nodes may show low latency in forming a network and during join and leave operation. But on the other hand the multicast cannot be controlled. The only way to restrict the service query or advertisement message is by specifying the Time To Live (TTL) parameter of the messages. Example of protocols that do not form an overlay network are GSD [2],

DSD [12], Konark [4], Splendor [8], Field theoretic approach [11] and protocols by Cheng and Marsic [1] and Varshavsky, Reid et. al. [9].

If we combine the existing ways of classifying the service discovery architectures, we get a classification that is a better representative of SDP architectures in MANETs. On this basis we categorize the SDP architectures in four categories, as given below.

- (i) directory-based with overlay support architecture
- (ii) directory-based without overlay support architecture
- (iii) directory-less with overlay support architecture
- (iv) directory-less without overlay support architecture

On the basis of above classification we give a brief overview of the SDPs in each of the category. One of the protocols in directory-based with overlay network support is Service rings [5]. It forms an overlay structure by grouping of nodes that are physically close and offer similar services. This overlay is formed on top of transport layer of ad hoc networks. The structure is called service ring. Each service ring has a designated service access point (SAP) through which the nodes within the ring can be accessed as it has all the information about the services offered within the ring. These SAPs are also connected with SAPs of other service rings thus forming a hierarchical structure. The directory information is kept in chosen edges that are dynamically selected. Similarly the protocol Lanes [6] also falls in the same category. It is inspired by Content Addressable Network (CAN) protocol, which is used for service discovery in wired peer-to-peer networks. Some nodes are grouped together to form an overlay network forming lanes of nodes. Each group is called a lane. Nodes in the same lane have the same directory replicated in each node cache. There are different lanes in a network that are loosely coupled with each other. The protocol given by Kozart and Tassiulas [7] forms a dominating set, also called virtual backbone from a subset of the network nodes. The nodes in the virtual backbones keep the directory, which stores the advertised information about other services in the network. The protocol given by Tyan and Mahmoud [10] forms cluster of mobile nodes in which each cluster has a gateway node. This gateway node is used for routing and keeping the directory.

The work in the field of directory-based without overlay network support is not much. Here we just have the protocol Splendor [8] by Zhu, Mutka et al. Even in this protocol the emphasis is on security aspect. It has four components, which are clients, services, directories and proxies. The directories are used for caching the service information and answering the client service requests. The proxies are used to authenticate the mobile services.

Similarly we have just one example of the work done in the directory-less with overlay network support architecture. Allia [3] is the only example. It follows a decentralized directory-less approach in which the nodes, which are geographically close form groups called alliances.

In the category of directory-less without overlay network support we have many SDPs as this architecture seems most obvious for the mobile ad hoc systems. For example the protocol by Cheng and Marsic [1] is directory-less P2P based on on-demand multicast routing protocol (ODMRP). GSD [2] and DSD [12] protocols are also P2P based and has a decentralized approach. Similarly an interesting approach is

a Field theoretic [11] suggested by Lenders, May et. al. This approach is inspired by electric field concept and uses a simple and distributed mechanism to find the best route to the closest service instance. It is totally a decentralized protocol without any central server. Konark [4] is also a completely distributed protocol and assumes IP connectivity between ad hoc nodes. Each device runs a stack of Konark application, SDP managers and registry. Another protocol that is directory-less and do not form overlay is proposed by Varshavsky, Reid et. al. [9]. It has two main components. A routing protocol independent Service Discovery Library (SDL) and Routing Layer Driver (RLD). SDL function is to store information about the service providers. RLD, which is closely coupled with the MANET routing mechanism, is used to disseminate service discovery requests and advertisements. Each node has the stack containing SDL and RLD to form a P2P networking or a directory-less architecture.

The categorization of protocols in different SDPs architectures is shown below.

Table 1. Categorization of SDPs Architectures

	Directory-based	Directory-less
Overlay	<ul style="list-style-type: none"> • Service Rings [5] • Lanes [6] • Kozart et al [7] • Tyan et al [10] 	<ul style="list-style-type: none"> • Allia [3]
No overlay	<ul style="list-style-type: none"> • Splendor [8] 	<ul style="list-style-type: none"> • Cheng et al [1] • GSD [2] • Konark [4] • Varshavsky et al [9] • Field Appr. [11] • DSD [12]

3 Management of service information

Service information includes any information about a service that is provided by a service provider in the advertisement. This information is used to describe, identify and discover a service in a network. The information may include name of the service, ID of the service, IP address of the service provider, port number of service point, protocol that server and a client may use to invoke a service [1], etc. The information about services must be stored somewhere so that other nodes can contact this node to discover a particular service. Management of service information include

aspects like where to store service information, time duration for which the information will be stored, distance in number of hops the information will travel as advertised by the service provider, etc. There are different ways different SDPs manage service information. Some protocols select a particular node among a group of nodes and store this information in a directory that resides on that particular node. Some store only on their local cache. The service information, depending on a particular advertisement policy, can be made to travel far or the advertisements can just be stored on neighbors. Some protocols require the service providers to refresh the service information before being deleted from the storage place. The exact mechanism how different SDP manages the service information is explained below.

First let us describe the mechanism of management of service information in protocols that do not have a centralized directory of storing service information. For example in Cheng and Marsic [1] protocol service information is stored on every node that is interested in the service. Each service provider multicasts advertisements about the services it can provide to the ad hoc network. Each server and its possible consumer make a multicast group. Any node that is interested in a particular service or services stores the advertisement in its local service registry and may send a service awareness reply to the service provider. Once some clients send back service awareness replies, the server sends its updated services advertisements by multicasting them to only those clients. With a similar idea in GSD [2] and Field theoretic approach [11] every node caches advertisement to maximum N hops (called the advertisement diameter). The service cache in each node thus contains a list of the local as well as remote services that this node has seen through advertisements. In this way each node in its local cache, contains the description of services that are within the advertisement diameter. By restricting the advertisement hops these protocols achieves better local memory utilization and also the probability of discovering a service in its vicinity. In Allia [3] the difference is that each node advertises its services only to the immediate neighbors, that is, the advertisement diameter is just one hop. Some nodes according to a local policy cache these advertisements. These nodes form an alliance with the advertising node. Thus every node stores, in its local cache, the advertisements from nodes in its alliances. As in Allia, DSD [12] also advertises its service information to all nodes in its radio range but here each node in addition to storing the service information may also forward the advertisement to other nodes, depending on the forwarding policy. Similarly in Konark [4] service information is stored on each node. This protocol has a special structure called service registry that is present on every device. It is used to store all the local service descriptions as well as the service descriptions that a node comes across through advertisements. In Varshavsky, Reid et. al. [9] scheme there is a service discovery library (SDL) on top of routing layer on every node. SDL maintains a service table that keeps record of service information.

The following explain the mechanism of management of service information in protocols that keep a centralized directory for storing the service information. For example Service rings [5] forms groups of nodes that are physically close to each other. Each group called “service ring” has a designated node called the “service access point” which keeps record of the services present in the ring. Similarly Tyan and Mahmoud [10] scheme form groups of mobile nodes based on their location. A node is chosen as a gateway that contains the directory. Lanes [6] also form group of

nodes called lanes in which each node knows its predecessor and successors. There is an anycast address of each lane in which all nodes share the same anycast address. The services advertised are sent to an anycast address of a lane. All nodes within a lane have the same directory replicated and thus have the same information stored. Kozart and Tassiulas [7] scheme is also not very different from the previous protocols as this protocol also selects a particular node for storing the service information. This protocol forms virtual backbone from some of the mobile nodes. When a particular node advertises its services, these are stored on the directories located on the virtual backbone nodes in case the service provider is itself not a backbone node. If the service provider is itself a backbone node then it registers services on the same node. Splendor [8] is a little different as in this protocol the service information is stored in special nodes called directories that are pre-assigned the task of storing service information and are not selected by the SDP.

4 Search methods

Searches can be used either to find the node having the directory or to find the services. The exact purpose of search depends on the type of storage method and the SDP architecture [15]. The search method depends on the type of network in which the search is made. Mainly there are two ways to search for information about the services available in the network.

- (i) The first method is used in networks that have directories for storing service information. The directory nodes keep information about the services available in a group and clients query these directories.
- (ii) The second method is used when there are no directories. Service providers advertise the services to all of the nodes. When a node is in need of a particular service, it searches its local cache for the presence of the service. If the service is not found query messages is sent to all nodes.

Service rings [5] is an example of SDPs in which searching is done using directories. It has special nodes called Service Access Points (SAP) that keep all the information about the services within the ring. When a node wants to search for a service, the query is routed through the ring structure, passing through SAPs of other rings and reaching only to those subrings that can possibly offer the service. This use of special overlay network with SAPs restricts the query flooding to only the selected nodes. Similarly in the protocol presented by Kozart and Tassiulas [7] the client forwards the service request to a virtual access point. These virtual access points, also called the Virtual Backbone Node (VBN) broadcast or multicast the query message to all the other VBNs. Only flooding the backbone nodes instead of all nodes in the network thus reduces the overhead of broadcasting a query. In Splendor [8] directories are first discovered by sending queries by the clients or the directories

themselves announce their presence periodically in the network. Clients after knowing the directory addresses, query the directories for services. In the SDP by Tyan and Mahmoud [10] the gateway of each cell provides the directory services containing information about the services of other gateways. When a client wants to search for a service, it sends a service request to its local gateway. The gateway searches its service advertisement cache and in turn gets a list of advertisements that corresponds to the service. In Lanes [6] the case is a little different. It has directories but the same directory is replicated throughout the lane overlay nodes. The service announcements are sent through a lane and service requests are sent through other lanes. These messages are sent through lanes by anycast routing.

The protocol given by Cheng and Marsic [1] does not have directories. When an appliance needs a service, it sends a query to service query multicast group. This group consists of a service provider and its possible consumers formed during the bootstrapping phase. In GSD [2] first the search is done in the local cache as it contains information about all the services within the advertisement diameter. When service is not found in the local cache then query request is selectively forwarded to a set of nodes based on semantic information. Similarly Allia [3] first checks the local cache for service information and if it is not available then active discovery is done by multicasting query request to the members of its alliance. If the service is still not available then the query request is broadcasted to other alliances in its vicinity. In Konark [4] and the protocol given by Varshavsky, Reid et. al. [9], the services are searched by first looking at the local cache and then, if not found, multicasting the service request to a fixed group of nodes in the network. Those nodes respond to the query message that can provide the service. In field theoretic approach [11] service advertisements are flooded through the network within a limited scope. Each node temporarily stores the advertisement and calculates the potential. When a client wants to search for a service, it forms a service query containing the service type of the desired service. This query is routed to the neighbor with a higher potential for that service, eventually reaching the service. In DSD [12] a service request based on ontology-based description is formed. The request is first matched with the services in the local cache and if services are not found, the request is selectively forwarded to other nodes based on the ontology descriptions. When the node does not have enough information to selectively forward a request, then a broadcast is made to the neighboring nodes.

5 Service Selection Methodologies

The query request from a client node to the network can result in many responses of matching services. Although there are many service discovery protocols that do not deal with the selection issue but for a service discovery protocol to be complete, handling of multiple responses of the same services should be taken care of and it should be part of the of service discovery protocol to select one of the available services for invocation. There can be different ways to select a service. For example it could be done manually or the selection procedure can be automated using some algorithm based on some criteria. The criteria or the metrics for service selection have

been defined differently by different protocols. For example the lowest hop count, current load of a service provider, bandwidth available of the communication channel between the service provider and the client, velocity of the service provider are some of the criteria.

Varshavsky, Reid et. al. [9] protocol integrates the service discovery and selection feature with the underlying routing protocol. They have demonstrated that proper service selection improves the overall network performance, by localizing the network communication. The mechanism used for service selection is simple. When multiple entries in the service table match the request, the client selects with the lowest hop count.

In Tyan and Mahmoud [10] proposal mobile agents are used for the service selection. When the mobile agents receive a list of advertisements from the service discovery phase, these agents move to different nodes while selecting the services according to some criteria. For example the user can specify the mobile agent to choose services with highest rating returned or services having some index values higher than user specified index value or the user can specify the mobile agent to just return the first available service.

In Field theoretic approach [11] client selects services using two metrics, one is the network distance, that is, the number of hops and other is the capacity of service (CoS). The algorithm for service selection is distributed and does not involve direct interaction with the client.

Splendor [8] specifies that the service selection will occur at client end but does not give detail of the algorithm used for service selection and also does not tell about the selection criteria. The protocols by Kozart and Tassiulas [7] and by Cheng and Marsic [1], GSD [2], Allia [3], Konark [4], Service rings [5], Lanes [6], DSD [12] do not tell any thing about selection mechanism.

6 Mechanisms for Mobility Support

In a system of mobile ad hoc network nodes keep on moving and changing their position with respect to each other. In a system in which the all of the nodes just keep information about their own services and not of the other nodes, mobility is not a big issue as searching is done by multicasting a query message to all nodes in the system. But the limitation of such a system is that it is not scalable. For example in [1] the mobility support is implicitly provided by the multicast mechanism. On the other hand, systems that:

- (i) have directory nodes that keep all of the information about other services in the network or
- (ii) have nodes that keep partial information about the services, for example services present in the neighbor node or
- (iii) form structured overlay networks

mobility is a real issue that has to be taken care by a SDP if the protocol has to function properly.

Mobility support implies that the information about services in the directory nodes is up-to-date under mobility. That is if a directory node is supposed to keep information about the all nodes in a group then it must have that correct information. If that node changes its position with respect to the group, the directory information should also be updated quickly. Only by this way the SDP can search the services in a timely manner. Mainly there are three different ways to support the mobility.

- (i) Updating service information
 - a. Event driven updating of service information
 - b. Periodical updating of service information
- (ii) Advertisement controls
 - a. Changing the rate of advertisement
 - b. Changing the diameter of advertisement
- (iii) Algorithms that maintain the structure of overlay network in SDPs that form structured overlay networks

Service information can be updated mainly by two ways. One is to update the service whenever there is any event occurring. For example when there is no route available to the service provider, the service information should be updated. The other way is to update the service information on regular basis for example by periodical advertisements as done by Konark [4], Splendor [8], DSD [12], Field theoretic approach [11], protocols by Kozart and Tassiulas [7] and by Tyan and Mahmoud [10]. The protocol by Varshavsky, Reid et. al. [9] uses both methods.

Some protocols change the rate and diameter of advertisements as the mobility of node changes. If the nodes are moving faster then rate of advertisement is increased and the diameter, that is the number of hops an advertisement can travel, is reduced. This type of mechanism is done in GSD [2] and Allia [3].

Some protocols form overlay structures and can only search correctly for services if that overlay structure is maintained. Due to mobility the overlay structure may get faulty. In this case there are special algorithms that try to maintain the overlay network structure. For example Service rings [5], Lanes [6], protocols by Kozart and Tassiulas [7] and by Tyan and Mahmoud [10]. A brief description of how mobility is managed in each protocol is given below.

Let us briefly describe how different protocols support mobility. The protocol by Cheng and Marsic [1] supports mobility by using multicasting for discovery of services in the networks. The mobility support is thus not explicitly provided by any special mechanism but it implicit in the multicasting the service requests [15].

In GSD [2] there are two parameters that can be adjusted for different mobility scenarios: the advertisement diameter and advertisement time interval. Advertisement diameter is the number of hops that an advertisement is expected to travel in the network and advertisement time interval is the time interval after which every nodes sends a list of services it has to all the nodes in its radio range. In high mobility scenarios, for example, the advertisement time interval can be reduced to cater for the rapidly changing vicinity. Similarly the advertisement diameter can be regulated with the dynamism of the network. In DSD [12], in addition to the mechanism discussed in

GSD [2], this protocol takes care of the effects of the mobility of nodes in the following way. The services announce when they enter the network and the neighbour nodes cache this information. If the advertisement is not refreshed after a specified time the information about the service will be removed from the cache of other nodes.

Allia [3] takes care of mobility by adjusting the advertisement rates and alliance diameter based on the mobility of the nodes. Regarding the advertisement rates one of the three methods can be employed. First is simply use a constant frequency rate for advertisements. This can be used for relatively stable networks. Second method is to use Multiplicative Increase Linear Decrease (MILD) algorithm or a Binary Exponential Back-off (BEB) Mechanism to vary the advertisement frequency. The advertisement frequency would be higher for more dynamic networks and low for less dynamic networks. Third possibility is sending out an advertisement only when it receives a new advertisement. The alliance diameter is the number of hops the advertisement may propagate in the network. Any node within the diameter would be able to cache the advertisement. For highly dynamic networks small advertisement diameter is adjusted and vice versa.

In Service rings [5] the overlay network is corrected which gets faulty due to the mobility of nodes. Each ring member only knows its successor and its predecessor. RingCheck messages, initiated periodically by the appropriate Service Access Points (SAP), circle through each ring to check its consistency. Every ring member receiving the message puts its predecessor information and forwards it to its successor. If a node does not receive such a message in one of its rings for a certain time it checks for a link breakage or a partition in the network. If any of these cases is detected then an appropriate algorithm is initiated to repair the ring.

In Lanes [6] the lane structure of the overlay network is maintained by different algorithms. Each node pings its upper neighbor and receives pings from its lower neighbor to maintain the lane. If any of the pings is missing either a node is detected to be vanished or the network is detected to be partitioned. In any case there are appropriate algorithms that are initiated to build a regular overlay structure according to the lane protocol specifications. Also there are algorithms for node logging in and logging off that keep the regular overlay structure.

In the protocol by Kozart and Tassiulas [7] the service registrations are done on periodic basis. In this protocol a virtual backbone is formed. To take care of frequent topology changes due to mobility or nodes vanishing, the dominating set feature of the backbone is maintained with the help of specific algorithms.

Splendor [8] and Konark [4] store service information as a soft state. When a service advertises itself, it also announces its lifespan. Before a service expires, it has to announce again. The proxies cache the information about the mobile services. Thus regular advertisements keep the information updated.

Due to mobility some of the service providers may not be accessible and some new ones may be in range. Reselection and rediscover are two mechanisms through which the protocol given by Varshavsky, Reid et. al. [9] takes care of the mobility of nodes. In reselection the services based only on the current entries in the service table are reconsidered. The policy when reselection should occur could be different. For example one reselection policy could be that reselection should occur when there is any change in the service table. Another policy could be to reselect the services when

there is no route to the server. In rediscovery the network is probed for up-to-date information about the available service providers.

In the protocol by Tyan and Mahmoud [10] mobility is supported by two mechanisms. First when a gateway node moves to another cell, it broadcast the service registry tree to the nodes in its previous cell. These nodes elect another gateway node. This gateway then starts using the service registry information. The second mechanism is by specifying time to live parameter, which is the physical clock time after which a service has to refresh its advertisement.

Field theoretic approach [11] protocol also uses periodical advertisements. The nodes can be disconnected from their neighbors due to mobility. This is determined by listening to the periodic update message from the neighbor node. If a node does not receive such a message for a long time it assumes a broken link and removes the neighbors from its table.

7 Service Description techniques

Service description is an abstraction of the facilities and characteristics of a service. The description of a service is necessary if it is to be utilized by other devices or services. The nodes in a network search for services by only looking at the descriptions of the services advertised by the service provider. A service, not properly described, may remain completely unknown to other devices in the network, thus defeating the objectives for which a service was formed. For these reasons SDPs usually describe the way services are described and the languages used for description. In MANETs SDPs we find three trends with regard to service description.

- (i) Most commonly used language for service description is eXtensible Markup Language (XML) and its extensions like DAML (DARPA Markup Language) [23] and Web Ontology Language (OWL) [24]. For example GSD [2], Konark [4], Service rings [5] and DSD [12].
- (ii) Some SDPs are independent of any description language. Any language or description method can be used in these protocols. For example one is free to use simple text attribute-value schemes or XML for describing services. For example Allia [3], Lanes [6] and the protocol presented by Varshavsky, Reid et. al. [9].
- (iii) In SDPs the issue of description is not discussed. These protocols are usually concerned only with the searching of a service and do not go into the details of other aspects of SDP. The authors by Kozart and Tassiulas [7], Zhu, Mutka et al in Splendor [8], Lenders, May et. al. in their Field theoretic approach [11] and Cheng and Marsic. [1] do not touch the issue of description of services in their protocols.

The details of describing a service in different protocols are following. Konark [4] protocol defines an eXtensible Markup Language (XML) based service descriptions. The description file is a plain text file that has all the information about the characteristic and functions of the service. GSD [2] use DAML (DARPA Markup Language) and OIL (Ontology Interference Layer) to define ontology to describe the services in mobile ad hoc networks. DAML + OIL is based on XML and the Resource Description Framework (RDF) [25]. The semantic capabilities of DAML make it a good choice for the description of services. The service requests are also expressed in DAML that are matched with the service description during the discovery process. The services are classified into groups based on class-subclass hierarchy present in DAML. The semantic features of DAML are used to reduce the network flooding. Web Ontology Language (OWL) is used in DSD [12] to describe services. OWL is also based on XML and RDF and is used in wired networks to describe services. The semantic class-subclass hierarchy present in OWL is used to described service groups. This also helps in selectively forwarding the service request.

In Allia [3] framework services can be described using any method, for example using XML or any other alternative. During the service discovery mechanism no description mechanism is specified, thus making Allia independent of any descriptions of services. Also the protocol presented by Varshavsky, Reid et. al. [9] is independent of any service description. To make the protocol [9] independent of any service description language, a matching of service advertisements by the service providers and service requests are handled by a pluggable matching module. The approach given in Lanes [6] is also independent of the service description. Similarly Service rings [5] will work with all the descriptions that satisfy the two conditions. First, there should be a distance function that allow to compare different service descriptions and second, there should be a summarize function which should produce a single new description if it is given a set of service descriptions. For example on simple taxonomies of services, both these functions can be defined. Another example is DAML-S language.

8 Conclusions and Future Work

In this paper we have surveyed SDPs for multihop MANETs. We selected twelve SDPs for MANETs and compared these protocols with respect to six important aspects. These aspects, which we chose for evaluating the protocols, are service discovery architectures, service information storage, search methods, service selection, mobility support and service description. There are many other aspects from which any SDP (for wired or wireless) can be evaluated, for example one of these is security, but these aspects are either not important with regards to MANETs or most of the protocols at present do not discuss these aspects. We have found a clear categorization that is a better representation of SDPs in MANETs. This categorization based on the service discovery architectures is given below.

- (i) directory-based with overlay support architecture
- (ii) directory-based without overlay support architecture
- (iii) directory-less with overlay support architecture
- (iv) directory-less without overlay support architecture

Most of the SDPs are in the category (iv), which seems natural for wireless mobile ad hoc networks. In spite of lot of research work most of the protocols are still in their initial phase of research and have only been verified using simulation studies. Very few have been implemented but just using a couple of devices. We strongly feel that there is a lot of potential in category (i), the directory-based with overlay support architecture for having scalable practicable real implementation of SDPs in MANETs. The reason being that in real world there are mobile nodes with varying degrees of mobility and with varying degrees of resources. A real world mobile ad hoc network may consist of mobile phones, PDAs, laptops and even we can include desktops, which are most of the time immobile and just can leave or enter a system. We observe an inverse relation between the mobility of a device and resources it has and the services it can offer. A mobile phone although less in resources or services to offer is much more mobile as compared to a laptop which is less mobile but has much more services to offer and also have lot of resources. Normally the protocols in the category (iv) consider all nodes having very few resources and therefore propose solution that does not pose any overheads on the protocols that is, having a directory-less without overlay support architecture. But this architecture has not been successful in providing a scalable and a practicable solution. Our position is that a more practicable solution for large scalable mobile ad hoc networks is only possible with directory-based and forming some sort of overlay structure. Presence of directory decreases the latency time for service discovery and service invocation. An overlay structure is helpful for having controlled multicast, thus helping in developing scalable protocol. SDPs with directories and also having an overlay structure clearly require more resources and may not be as lightweight as SDPs in category (iv). We can get rid of these limitations if we also include nodes that have more resources like laptops and even desktops (which although are not mobile but can be included in ad hoc category as they can join and leave the system).

We found that there are not many protocols that discuss the security aspect of SDP. Any SDP if it has to be practicable cannot ignore the security aspect. This is another area of research that can be pursued in the domain of SDP for MANETs.

Mobility is an important dimension in SDP. We found that there are mainly three ways that are used to handle mobility. These are:

- (i) Updating service information
- (ii) Advertisement controls
- (iii) Algorithms that maintain the structure of overlay network

Most of the protocols use either one of these methods. We think this is another area of research that can be probed into for finding ways to improve the mobility support by using some intelligent technique based on all these three methods and even some other method.

Service discovery is an important and an active field of research. Especially in the domain of Mobile ad hoc networks, which is also a very active field of research, the importance of service discovery protocols is even more. Still there are many open problems that need to be addressed before SDPs can be made practicable.

References

1. L. Cheng and I. Marsic. Service discovery and invocation for mobile ad hoc networked appliances, December 2000.
2. Dipan Chakraborty, Anupam Joshi, Tim Finin and Yelena Yesha. GSD: A novel group-based service discovery protocol for MANETs. In 4th IEEE Conference on Mobile and Wireless Communications Networks (MWCN). IEEE, September 2002.
3. Olga Vladi Ratsimor, Dipanjan Chakraborty, Anupam Joshi, Timothy Finin. Allia: Alliance-based service discovery for ad hoc environments. In ACM Workshop on Mobile Commerce WMC'02, September 2002.
4. S. Helal, N. Desai, V. Verma and C. Lee. Konark- a service discovery and delivery protocol for ad hoc networks. Proceeding of the Third IEEE Conference on Wireless Communication Networks WCNC, March 2003.
5. Michael Klein, Birgitta Konig-Ries and Philipp Obreiter. Service rings – a semantic overlay for service discovery in ad hoc networks. In DEXA Workshops, pages 180-185, 2003.
6. Michael Klein, Birgitta Konig-Ries and Philipp Obreiter. Lanes – a light weight overlay for service discovery in mobile ad hoc networks. Technical Report 2003-6, University of Karlsruhe, May 2003.
7. Ulas C. Kozart and Leandos Tassioulas. Network layer support for service discovery in mobile ad-hoc networks. Proceeding of IEEE/INFOCOM-2003, April 2003.
8. Feng Zhu, Matt Mutka and Lionel Ni. Splendor: A secure, private and location-aware service discovery protocol supporting mobile services. Proceedings of the First International Conference on Pervasive Computing and Communication PerCom'03, Pages 235-242, 2003. ACM Press.
9. Alex Varshavsky, Bradley Reid and Eyal de Lara. A cross layer approach to service discovery and selection in MANETs. January 2004.
10. Jerry Tyan and Qusay H. Mahmoud. A network layer based architecture for service discovery in mobile ad hoc networks. CCECE 2004, May 2004 IEEE.
11. Vincent Lenders, Martin May and Bernhard Plattner. Service discovery in mobile ad hoc networks: A field theoretic approach. Pervasive and Mobile Computing 2005.
12. Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha and Tim Finin. Toward distributed service discovery in pervasive computing environments. IEEE Transactions on Mobile Computing, February 2006.

13. Toh, C.K., "Ad Hoc Mobile Wireless Networks. Protocols and Systems", Prentice Hall PTR, New Jersey, 2002, pp. 231-242.
14. Chunglae Cho and Duckki Lee. Survey of service discovery architectures for mobile ad hoc networks. Unpublished 2005, Computer and information sciences and Engineering Department, University of Florida Gainesville, USA.
15. Raluca Marin-Perianu, Pieter Hartel and Hans Scholten. A Classification of service discovery protocols. Unpublished June 2005.
16. Feng Zhu, Matt W. Mutka and Lionel M. Ni. Service discovery in pervasive computing environments. IEEE Pervasive Computing, October 2005
17. Jini Technology Core Platform Specification, v. 2.0, Sun Microsystems, June 2003 www.sun.com/software/jini/specs/core2_0.pdf.
18. UPnP Device Architecture 1.0, UpnP Forum, Dec. 2003 www.upnp.org/resources/documents/CleanUPnPDA10120031202s.pdf
19. Salutation Architecture Specification, Salutation Consortium, 1999
20. E.Guttman et al., Service Location Protocol, v. 2, IETF RFC 2608, June 1999 www.ietf.org/rfc/rfc2608.txt
21. M. Nidd, "Service Discovery in DEAPspace," IEEE Personal Comm., August 2001, pp. 39-45.
22. Specification of the Bluetooth System, Bluetooth SIG, Feb. 2003.
23. <http://www.daml.org>
24. <http://www.w3.org/TR/owl-features/>
25. <http://www.w3.org/RDF/>

A Fault Tolerance Support Infrastructure for Web Services based Applications

Nicolas Salatge and Jean-Charles Fabre

LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 04 – France

{nsalatge, fabre}@laas.fr

Abstract: *Web Services provide a new approach for the development of distributed applications, namely the Service Oriented Architectures (SOA) based on “service” contracts. Currently contracts do not address in deep dependability aspects of Web Services implementation. In addition, Web Services providers do not pay very much attention to dependability attributes, their objective being to maximize the number of customers for non-critical applications. As far as more critical applications are concerned, dependability mechanisms are highly required. The support infrastructure proposed in this paper enables both clients and providers to add dependability mechanisms to web services used to build large scale applications. As this approach provides separation of concerns, such dependability mechanisms can easily be adapted to the needs. To this aim, a dedicated language (a DSL, Domain Specific Language) has been designed to simplify the description of dependability mechanisms and make them more robust. We show that this approach enables the implementation of adaptive dependability mechanisms. A platform for implementing dependable applications based on Web Services has been developed (services and tools) and used with various Web Services on the Net.*

Keywords: Web Services, Service Oriented Architectures, Domain Specific Language, Fault tolerance.

1 Introduction

Service Oriented Architectures (SOA) [1] enable the development of loosely-coupled and dynamic applications. Such applications are based on a core notion, the notion of service, and on a contract linking a client and a service provider. This type of architecture is currently used for large-scale applications like e-commerce, but should be of interest in the future for applications having stronger dependability requirements.

Today, Web Services are the only way available to realize service-oriented applications. From a pure “marketing” viewpoint, Web Services are developed to satisfy client needs from a functional viewpoint, to be easy to maintain, and also to provide some high level of quality of service. Web Service providers must also take care of the reliability and availability of their individual Web Service implementation. However, the provider cannot take into account all possible client needs and constraints for the development of a given application. This means that additional mechanisms must be developed and tailored for a particular context of usage.

This is exactly the kind of problem we tackle in this paper. Application developers look at Web Services as OTS (*Off-The-Shelf*) components and thus they ignore their implementation and their

behavior in the presence of faults. In a sense, clients need a support infrastructure to implement fault tolerance mechanisms that can be dynamically attached to a given Web Service. The same Web Service can be used in several service-oriented applications and thus with different dependability constraints by different clients. In addition, a given client may want to apply different fault tolerance strategies regarding a given Web Service over the lifetime of its application.

To this aim, we propose a framework to help clients making so-called *Specific Fault Tolerance Connectors* (SFTC) that implement filtering and other robustness and error detection techniques (e.g. runtime assertions) together with recovery mechanisms that are triggered when the WS does not satisfy anymore the dependability specifications (see figure 1). The same Web Service can be used in several service-oriented applications with different dependability constraints and thus taking advantage of several connectors.

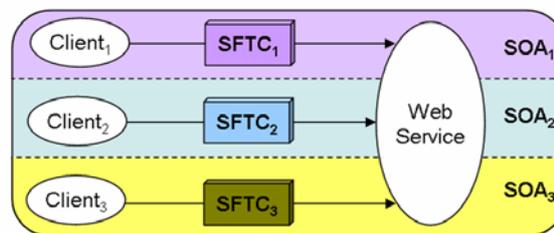


Figure 1: The Specific Fault Tolerance Connectors concept

The problem is indeed similar to the use of COTS components in safety critical systems, and previous work showed that mechanisms like fault containment wrappers were a possible solution [2]. In the SOA context, the objective is to satisfy all possible clients' dependability needs. The approach must be more adaptive and enable dependability mechanisms to be 1) defined on a case-by-case basis and 2) highly dynamic, i.e. changed according to the needs. To this aim we propose:

- 1) a language support (DeWel) to describe the dependability features of a connector and,
- 2) a support infrastructure to dynamically manage and run connectors in real applications.

The paper is organized as follows. Section 2 summarizes our motivations and the basic principles of the work. Section 3 describes the organization of the support infrastructure to develop, execute and manage user-defined connectors to unreliable web services. Section 4 describes the main features of the language defined to make reliable specific fault tolerance connectors. Section 5 focuses on connector provided error detection mechanisms, whereas Section 6 discusses the recovery strategies. Section 7 summarizes our first experiments. Related work is addressed in Section 8 and section 9 concludes the paper.

2 Motivations and basic principles

As far as dependability is concerned, we have to consider the failures that can occur in operation, and which can be due to a number of different types of faults. Just to be convinced, we have performed some experiments with different Web Services during a short period of time (72 hours) and we observed that some failures are quite frequent (see Table 1). The observed failures are mostly due to transient interaction faults and do not reflect the QoS of each target service. Consequently, the observed failure rate can change very much over a long period of time. This shows the kind of impairments to consider when developing large-scale applications on top of Web Services.

	Number of request	Type of errors									
		Number of error	(502)Bad Gateway	Could not translate. Network problem.	java.lang.NullPointerException	Connexion refused	Re-initialized connection by the correspondent	(404)Not Found	Connection closed: expiration of the waiting period	No access path to reach the target host	Service hang
BabelFish	3423	2	0	2	0	0	0	0	0	0	0
FedEx	3406	3	0	0	3	0	0	0	0	0	0
Google	3308	28	28	0	0	0	0	0	0	0	0
MSN Search	3327	0	0	0	0	0	0	0	0	0	0
Amazon – US	2350	0	0	0	0	0	0	0	0	0	0
FraudLabsWebService	3501	1	0	0	0	1	0	0	0	0	0
Temperature ConvertService	3362	260	0	0	7	8	2	202	40	1	0
TimeService	178	1	0	0	0	0	0	0	0	0	1

Table 1: Example of Web Services errors collected over a 72-hour period

Companion works [3, 4] showed that many faults can impair Web Services in operation. Beyond basic physical faults that may affect the nodes running the services, it has been observed that communication faults was a significant source of errors in large scale applications on the Net [5, 6]. More importantly, due to the complexity of the WS multi-layer runtime support, software faults have to be considered as a first class type of problems. Among the various software components and just to give an example, the SOAP parser has a prime importance and can be subject to development faults like incorrect parsing of request messages or mapping of data types, bad catching of error codes returned by the operating system. In summary, the development of critical applications over Web Services must take care of many fault sources:

- 1) Physical faults affecting the computers and the networking hardware infrastructure;
- 2) Software faults affecting the software components of WS runtime support (OS, application servers, SOAP engines, etc.);
- 3) Evolution faults related to inconsistencies between current WSDL versions and existing stubs generated from older versions;
- 4) Interaction faults dealing with the service access point, like access point unreachable, non existent or changed;

- 5) Communication faults leading to message lost, duplication and/or omission;

Our idea is first to provide the user with means to equip individual WS with customized and efficient error detection mechanisms, i.e. to transform a WS into a self-checking software component [7, 8]. In a second step, we propose some partially built-in replication techniques to perform error recovery. Depending of the coverage of the fail-silent assumption of the self-checking WS, the recovery procedure can range from simple switch to a spare component to error masking strategies.

The core element of the proposed framework is the notion of user-defined fault tolerance connector between clients and providers. The user can be a client, a web service provider or any third party user interested by dependability mechanisms. A connector is defined as such:

- A connector is a software component able to capture Web Service interactions and able to perform partially built-in fault tolerance related actions.
- Its role is two-fold (see Figure 2): it performs i) runtime assertions by applying checks to input/output requests for error confinement and ii) recovery actions for fault tolerance, according to a given failure model and depending on state management features of the target WS.

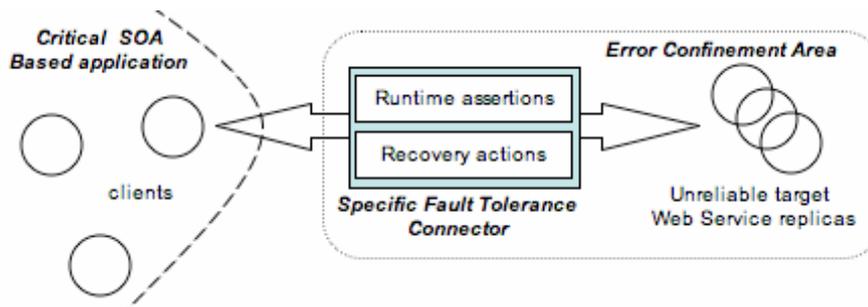


Figure 2: Role of Connectors

The dependability features that can be implemented in this way depend on several parameters, including the specific features of the Web Service that is under control. Checking interaction might be different for a client who aims at protecting himself from corrupted information returned from a service, or conversely, for a provider to protect the service against erroneous clients. To describe these different and specific fault tolerant mechanisms, we propose a *Domain-Specific Language* (DSL) named DeWel (Dependable WS language). DeWel aims at making more reliable the development of specific fault tolerance connectors between clients and providers on a case-by-case basis. This is addressed in Section 4.

Regarding recovery, it is very much dependent on the nature of the provider, stateful or stateless, and obviously on the considered fault model. Although such fault tolerance mechanisms can be tailored to the target service, it is however possible to design an overall framework providing separation of concerns between applications and dependability mechanisms. This is discussed in Section 6.

3 Overview of the IWSD framework

The management and the execution of connectors rely on a specific platform that is a third-party infrastructure between clients and providers. The storage, look-up, delivery, loading and execution of Specific Fault Tolerance Connectors are supported by the IWSD platform, an *Infrastructure for Web Services Dependability*. The framework provides fault tolerance mechanisms without being intrusive both for clients and providers of Web Services. Connectors are generated by the DeWel compilation tools suite (i.e. as a dynamic library) and stored into the SFTC Repository of the platform. At runtime, the platform provides support to perform the fault tolerance mechanisms requested by the user and implemented as a Specific Fault-Tolerance Connector.

The IWSD platform (see Figure 3) is composed of the following services:

- **The Dependability Server (DS)** is responsible for the interception of SOAP messages directed to a given WS, the loading and the execution of the corresponding Specific Fault-Tolerance Connector. The DS can be perceived as a sort of virtual machine running connectors, and includes an authentication module, HTTP and SOAP parsers and a loader of connector;
- **The Management Server (MS)** provides i) the compilation tool suite of DeWel programs and the storage of corresponding connectors into the SFTC Repository, ii) the management of user accounts and configuration information (e.g. communication endpoints for a given service);
- **The Health Monitor (HM)** is in charge of collecting all failure information and error reports from each component to evaluate the current level of dependability of the **Dependability Server**. The Health Monitor is also in charge of supervising the runtime status of Web Services in operation by collecting errors reported by the active connectors.

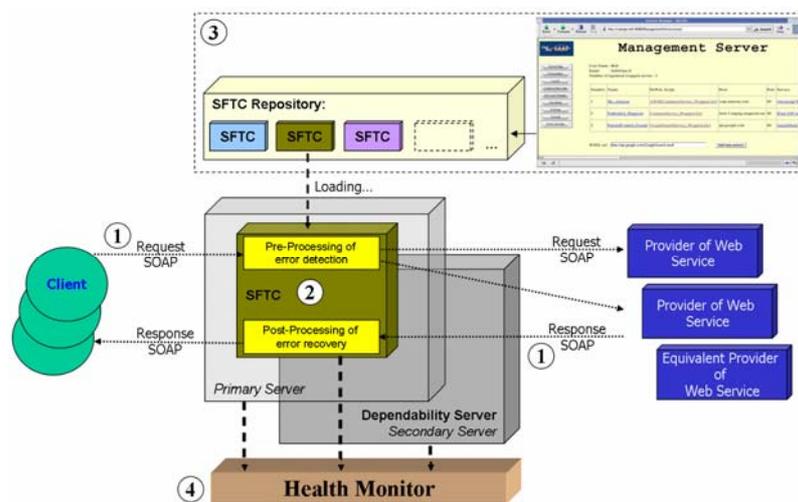


Figure 3: IWSD, an Infrastructure for Web Services Dependability

As shown in Figure 3, the **Dependability Server** itself can be made fault tolerant using conventional techniques, for instance using a duplex architecture to tolerate crash faults or any other strategy to improve its reliability and availability. As we consider world-wide applications based on web services, many instances of *Fault Tolerant Dependability Servers* should be available on the Net. Making the

DS highly dependable is attained using conventional fault tolerance techniques and is thus out of the scope of this paper.

4 A language support to implement connectors

The notion of the connector is the corner stone of the approach developed in this work and can be perceived as a single point of failure in the overall architecture. This means that its development must target reliable code and its support infrastructure must ensure reliability in operation. In other words, the connector must be a self-checking component running on top of a fault tolerant platform.

4.1 Specification of the language

The development process of the connector must prevent faults to be introduced. This can be achieved by several means, and in particular using the principle of *Domain Specific Languages*. The design of the DeWeL language should have two major objectives:

- prevent software faults that usually appear in operational software using static (compilation) and dynamic (on-line) verification;
- provide a finite set of language construct to declare recovery strategies and write runtime assertions.

Field software development faults have been classified by several industrial companies (e.g. IBM, HP, etc.). For instance, in ODC (Orthogonal Defect Classification) from IBM [9], fault types are classified as follows: assignments, incorrect data/parameter checking, algorithm correctness problems, timing/serialization of resources, incorrect functions. This kind of faults can be prevented at least partially by reducing the possibilities of the language in terms of data types and algorithmic constructs. This is exactly what is enforced by design and coding standards for safety critical software, like CENELEC 50128 for railways (*i.e.* “*avoidance of language constructs like dynamic objects, dynamic data, recursion, pointers, exits etc.*”) or recommended in the Software Code Standards section of the DO-178/EUROCAE standard for avionics. This justifies the restrictions imposed by the language. In spite of these restrictions, the basic language features must be used to realize a connector:

- *Defensive programming*: The first objective is to make sure that the user is able to write assertions on input and output requests, i.e. the restrictions do not impair the expressiveness.
- *Exception handling*: communication errors and service errors must be captured and processed within the connector using the restricted expressions allowed by the language or forwarded back to the client for processing at an upper abstraction level.
- *Recovery strategy selection*: Regarding recovery strategies only declarative statements are allowed; within a connector, the user sees recovery strategies as built-in software components that can only be parameterized. The implementation of these “*partially*” built-in strategies is discussed

in section 6. The errors detected by assertions and exceptions trigger the recovery procedure if any.

Language restrictions	Specific features to be used instead	Error avoidance	Checked property
- No dynamic allocation - No pointers		Segmentation fault, Not enough memory	Resources and memory control
- No files	A fixed size log file attached to a connector	Not enough memory	
- No indexed access to arrays	Controlled loop (foreach)	Table overflow	
- No standard loops (while, for)		Service hang	Termination
- No functions - No method overriding	- pre-defined functions - specific objects methods		
- No recursive construct			
- No external access to other users data space or system resources		Data corruption	Non-interference

Table 2: Essential characteristics of DeWel

Table 2 shows the restrictions enforced in DeWel, the type of errors they prevent, and the type of critical property targeted by this DSL. Some specific recommendations are proposed to overcome such restriction in practice, i.e. specific programming features to enhance connectors' robustness. Thanks to such restrictions, it is possible to perform efficient static verification at compile time and provide automatic code generation including the dynamic verification of SOAP messages parameters.

4.2 Template of a DeWel program

To avoid learning a new language and, more importantly, to simplify its use and understanding, DeWel borrows its syntax from C (for conditional actions, arithmetic and logic expressions, etc.). However, DeWel differs very much from general purpose programming languages like C. It can be seen as a declarative language, in particular regarding recovery strategies that are only parameterized, and as a restricted imperative language for the expression of runtime assertions. The later correspond in pre and post conditions that encapsulate the execution of the service (like Before, After, Around advices in Aspect Oriented Programming [10]). As shown in Figure 4, a DeWel program is in fact developed from a pre-defined template composed of several sections:

- Declaration and parameterization of the RecoveryStrategy selected;
- Definition of assertions:
 - o Pre-Processing assertions to check the validity of input requests;
 - o Post-Processing assertions to check the validity of WS responses;
- Definition of error handlers:
 - o CommunicationException handlers processing communication errors;
 - o ServiceException handlers processing the target Web Service errors;

The template is automatically generated from the WSDL document describing the target Web Service. All service operations using the SOAP protocol and described in this document are imported into the template (see Figure 4). In practice, the pre-processing assertions check the validity of some input

parameters and the post-processing assertions filters both unacceptable replies and, for instance, upper bounds on return attributes of SOAP messages¹.

```

SpecificFaultToleranceConnector myConnector extend myService.wsdl {
  type paramR operation1(type param1, type param2, ..., type paramN) {
    RecoveryStrategy:
    // Definition of Recovery Strategy
    Pre-Processing:
    // Assertion on service request
    return operation1;
    Post-Processing:
    // Assertion on service response
    return operation1;
    CommunicationException ce:
    // Communication error handler
    return ce;
    ServiceException se:
    // Service error handler
    return se;
  };
  type paramR operation2(type param1, type param2, ..., type paramN) { ... };
  ...
  type paramR operationN(type param1, type param2, ..., type paramN) { ... };
}

```

Figure 4. Example of a DeWel Program Template

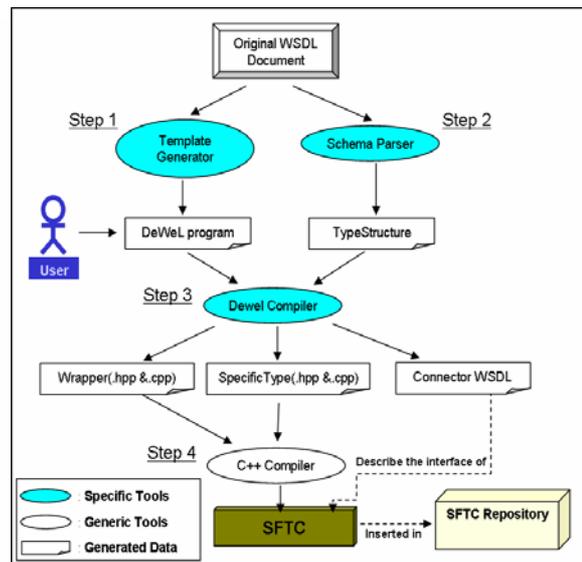


Figure 5. Code Generation Process

4.3 Code Generation Process

This section describes the code generation process that produces Specific Fault Tolerance Connectors from WSDL contracts. As show in Figure 5, this code generation process is composed of four steps, each of them supported by a tool. Four tools contribute to the implementation of this code generation process, and three have been developed on purpose. These tools are involved in the three first steps of the process to produce robust connectors:

Step 1 – WSDL Document Analysis and Template Generation: First, the validity of the WSDL document is checked together with the internal operations provided by the Web Service. A DeWel template contains the signature of Web Service operations extracted from the WSDL document. The user can then insert appropriate fault tolerance mechanisms in the corresponding sections of the template (recovery strategy, pre-processing, post-processing, communication and service exception handlers). This step is assisted by the Template Generator tool.

Step 2 – XML Schema Analysis and type generation: This second step is only required when the WSDL document includes specific XML schema to describe data types defined by the Web Service. In this case, a particular data structure (named TypeStructure) is produced to simplify the manipulation of such data types in the template. This type structure is the result of the translation of complex XML types into DeWel types (e.g. C++ style). These data types are used to perform checks on service request parameters in assertions. The Schema Parser tool has been developed for this step.

¹ Dewel programs can be found at the url: <http://www.laas.fr/~nalatge/IWSD> (DeWeL examples)

Step 3 - Compilation of DeWel programs: The compilation process has two inputs: (i) the template that has been updated by the user with fault tolerance mechanisms (recovery strategies and assertions) and (ii) the TypeStructure provided by the Schema Parser. The compilation verifies the consistency of the user's source code and the data type information contained in the TypeStructure. The output of the compilation process is a set of C++ files corresponding to the user-defined fault tolerance mechanisms of the connector and the specific data types of the service. The DeWel Compiler that is currently available is responsible for this step.

Step 4 - Creation of a connector as a dynamic library: This is the last step of the process in which a conventional C++ compiler is used to produce the final SFTC that will be used at runtime. A dynamic library is obtained from the C++ files generated in step 3. The connectors are then registered into a repository of the platform. This step is assisted by the GNU/C++ compilation suite in our case.

4.4 Concluding remarks on language issues

Together with its IWSD supporting infrastructure, the aim of the language is to simplify the implementation of reliable connectors in order to make error confinement areas around unreliable Web Services. Reliable connectors can be developed by a user thanks to a template extracted from the Web Service WSDL document. As a consequence, a new service is obtained with better and, more importantly, user-defined fault tolerance features. This enhanced version is described as a new WSDL document and a new access point. This new access point is registered into IWSD. Each connector is thus visible as a WSDL document on the Web².

Some recent works have used advanced reflective language features to include non-functional mechanisms into the implementation of Web Services (in [11] and [12]). This was performed with Java and AspectJ. To do this, the Web Service must be implemented in Java and the code must be available. The main advantage of DeWel is its independence with respect to WS implementation details, only the WSDL document is needed. Its supporting platform IWSD is independent from any client or provider and thus can be easily inserted in any application.

5 Error detection mechanisms

The IWSD framework provides an effective support for programming and executing Web Service Connectors for fault tolerance. In this Section we discuss the different kind of connectors and some aspects of their behavior in operation. A comprehensive description of the connectors in action can be summarized as follows. Provided the pre-processing assertion is passed, the target WS executes and produces output results. When an error is detected by the post processing assertion or when an exception is returned to the connector, two cases are possible:

² The WSDL document of the connector for Google is available to the following URL: "<http://www.laas.fr/~nsalatge/IWSD/>" (see WSDL connector of Google).

- When no recovery procedure is defined in the connector, then an exception is returned to the client.
- When a recovery procedure is defined, an exception is returned to the client only if all spare replicas have been used and the recovery procedure fails.

5.1 Assertions

The Connector foresees two types of assertions, *implicit assertions* that are automatically performed by a tool, or *explicit assertions*, which are user-defined thanks to the DeWel language. The implicit assertions are done by default when a message (request or response) is analyzed by request parsers that perform syntactic verifications (e.g. badly formed or corrupted SOAP messages).

The explicit assertions are tailored to each operation of a service and written by the user in DeWel. They are able to control for instance the validity of a typical data value. User-defined assertions can be much more complex and based on some formal expressions composed of several parameters. It is worth noting that assertions can take advantage of connector local variables to keep track of the some history of the WS connection. Some assertions can thus be implemented using this recorded information. The violation of such assertions can thus signal an error to the client by means of exceptions.

5.2 Exceptions

Some specific high-level exceptions are strongly related to the proposed framework in operation. They can be also implicit or explicit. For instance, an exception can be returned when a SOAP request cannot be analyzed by the parsers (implicit exception). A specific error defined by the user in the pre and post-processing assertions may also lead to an exception (explicit exception). Two exceptions mechanisms are provided to the user:

- **Generation of an exception.** The user can define and generate an exception when an assertion is false. A user-defined exception can be directly returned to the client, e.g. `return SOAPException ("Invalid Value");`
- **Catch of an exception.** The user can catch and process two types of exceptions: *Communication Exceptions* and *Service Exceptions*. The processing of the exceptions is defined in the corresponding sections of the DeWel program template (see. Figure 4).

A *Communication Exception* is returned when the request cannot be sent by the *Dependability Server* to the targeted Web Service. A *Service Exception* is returned when the targeted service returns an error as a SOAP message. Within the connector, the catch of an exception is implicit but the handler code to process the exception can be complex and defined by the user. As an example, a very simple exception handler can just be the logging of error information for later analysis.

6 Recovery Strategies and replication

The principles of Web Services naturally provide possible replicas on the Internet since several implementations of the same services (i.e. same WSDL document) can be available. This natural redundancy can be used to improve the dependability of an application based on Web Services. One can think that some similar services can also be found. Similar means that they can provide an acceptable service instead the original one, a sort of degraded service (see section 6.1). Various replication strategies can be used depending of the nature of the target WS and the dependability requirements of the client application (see section 6.2). However, by definition replication means that state management cannot be ignored (see. Section 6.3) which is a difficult issue with *OTS* components.

6.1 Various sorts of replicas

We have thus to consider two types of services which can be used to implement recovery mechanisms. First of all, **Identical services** correspond to a unique WSDL document, but the access point is different (e.g replicas for Amazon: JP, US, FR, ...etc.). Different implementations of the service help to tolerance transient faults of the WS runtime support but also design faults of the WS. A simple switch to a different replica is done in this case.

More importantly we have to consider so-called **Equivalent Services**: the WSDL documents are different but can be considered as providing a similar specification of the original service. In order to take advantage of **Equivalent Services**, we introduce the notion of **Abstract Web Service (AWS)**. An **AWS** does not have any functional reality but have a WSDL document; it is an abstraction of several similar services. The connector associated to an **Abstract Web Service** must convert so-called **“abstract requests”** to concrete requests and vice-versa for the responses. To create such connector for equivalent services A and B, we have to consider four types of parameter (see. Figure 6):

- **Surjective parameters (★)**: A parameter P_A of service A is said to be surjective when there is at least one parameter P_B in service B so that there exists a transformation function « strans » that satisfies « strans (P_A)= P_B ».
- **Injective parameters (□)**: A parameter P_A of service A is said to be injective when there is at most one parameter P_B in service B so that there exists a transformation function « itrans » that satisfies « itrans (P_B)= P_A ».
- **Bijective parameters (○)**: a bijective parameter is both surjective and injective. It signifies that a parameter of the service A has one only associated parameter in the service B (for example, the query string for search engine service such as Google or MSNSearch).
- **Local parameters (●)**: these parameters are proper and mandatory to access one specific service. There is no possible mapping to convert a local parameter of service A to a local parameter of service B (for example, authentication parameters).

An **Abstract Request** is not the union of concrete requests. We can define an abstract request as composed of all necessary and sufficient parameters to derive a request for concrete services, namely A and B in the example. The minimal abstract request is simply defined as composed of:

- 1) Bivariate parameters belonging to **only one** concrete request (A or B);
- 2) Surjective parameters belonging to **all** concrete requests (A and B);
- 3) The local parameters belonging to **all** concrete requests (A and B).

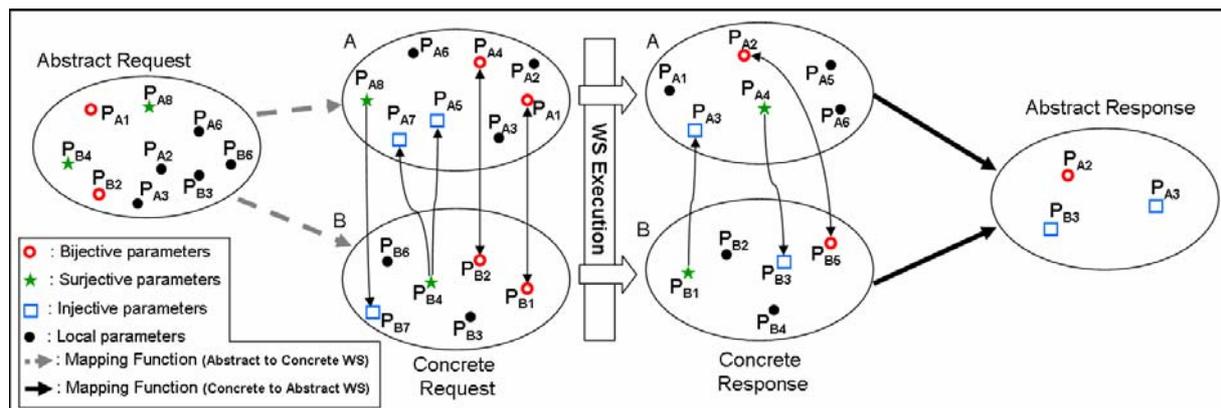


Figure 6: Equivalence of WS

It is worth noting that surjective parameters enable retrieving injective parameters of all concrete requests. For instance, in the example of Figure 6, P_{B7} is an injective parameter of service B that can be retrieved from the surjective parameter P_{A8} belonging to the abstract request, provided the strans function is known: $\langle\langle \text{strans}(P_{A8}) = P_{B7} \rangle\rangle$.

Conversely, an **Abstract Response** can be defined in the same way. Only bivariate and injective parameters belong to abstract responses, surjective parameters are transformed into injective parameters, provided the itrans function is known: $\langle\langle \text{itrans}(P_{B1}) = P_{A3} \rangle\rangle$.

These abstract request and response represent one operation of the **Abstract Web Service** and so, can be defined in a WSDL document. The aggregation of transformation functions (“iTrans” or “sTrans”) associated to each parameter of the service corresponds to the mapping functions necessary to convert an abstract request to a concrete service request (dashed arrow) or a concrete response to an abstract service response (bold arrow)³. From a technical viewpoint, these mapping functions are XSLT scripts [13] which contain node transformations on SOAP requests and response.

The above simple solution to address the equivalence of services on the Net can take advantage of more advanced research on *Ontologies* whose purpose is to add semantic to Internet resources and to classify them [16, 17]. This approach also called *Web Semantic* [14] should be able to help searching and classifying similar Web Services. Recent works have also addressed the matching between WS

³ The WSDL document of the Abstract Service for Google and MSN as well as mapping and unmapping scripts are available to the following URL: "<http://www.laas.fr/~nsalatge/IWSD/>" (see WSDL connector of Generic Search Engine).

[15-17] just by searching similarities (e.g bijective, surjective or injective parameters) or incompatibilities (e.g local parameters) between WS operations.

6.2 Replication strategies

The DeWel section entitled **RecoveryStrategy** enables the user to select an appropriate replication strategy for a target service. It is clear that fully implemented mechanisms cannot be provided. The provided mechanisms are partially built-in essentially because of state management issues. The final fault tolerance strategy to be used is the result of collaboration between the client, the connectors supported by the ISWD platform, and the provider when possible.

The recovery strategies are defined using the private functions given in the table 2. For each DeWel function, this table summarizes the principles of the available recovery strategies and the role of each participant (connector, provider and client). Cloning a replica is not considered here.

DeWel functions	Recovery strategy	Connector role vs. recovery strategy	Provider role	Client role
1) BasicReplication	Passive Replication with no state management	Switch to another replica	State recovery management	
2) StatefulReplication	Passive Replication with WS state management using provider-defined state management functions	Checkpointing using provider-defined state management functions Switch to another replica	Provision of state management functions	
3) LogBasedReplication	Passive Replication with state management using session mechanism (log and repetition)	Logging of all the requests during a session. Re-execution of logged requests on another replica		Invocation of Session Start and Session End.
4) ActiveReplication	Active Replication without vote	Multicast requests to the set of replicas. Selection of the first acceptable response		
5) VotingReplication	Active Replication with vote	Multicast requests to the set of replicas. Performs the vote or a decision procedure among multiple responses		

Table 3: Recovery mode

Passive Replication strategies (*BasicReplication*, *StateFulReplication*, *LogBasedReplication*) involve sending the request to only one replica that processes the request. In case of errors detected (by post-processing assertions or exceptions), a spare replica is used to perform the request. The main difference between these strategies resides in the way state management is performed as discussed in the next section. In this case, beyond the execution of assertions, the connector provides the routing and failure detection (unreachable primary because of *node crash*, *service crash*, *service hang*).

Regarding **Active Replication** strategies (*ActiveReplication*, *VotingReplication*), the connector multicasts the request to N Web Service replicas when the pre-processing assertion is passed. Currently, two configurations are available:

- **No voting** : the connector receives the N responses from the replicas, and sends to the client the first response which matches the post-processing assertion. This mechanism tolerates *node crash*, *service crash* or *service hang*. In practice, active replication has been used in our experiments with all registered *Identical Service* replicas of Amazon (see section 7).
- **With voting**: the recovery mechanism with (majority) voting can tolerate value faults (*data corruption*). The connector receives responses provided by $2f+1$ service replicas, up to f faults can be tolerated in this case. A library of voting algorithm variants (bitwise, average, median, etc.) can be provided to the user for the configuration of the vote in the connector.

These recovery strategies can be customized by the user, i.e. they can be parameterized or variants can be derived from the existing ones (e.g. inspired by approaches like *Recovery Blocks*, *N-Version Programming*, *N-Self-Checking Programming*). Clearly, all of them have a list of replicas as a first parameter. Other parameters concern timer values, checkpointing variants, decision functions, etc. More advanced strategies can also be created to tackle other non-functional aspects, including application dependent recovery strategies.

6.3 WS State Management

When the provider is responsible of the state management (see row 1 on Table 2) the role of IWSD is limited, but at least very useful for stateless WS. *BasicReplication* provides redirection mechanisms, failure detection being done by means of assertions and exceptions. The delay to switch from one replica to another is defined by the provider as a parameter. It corresponds to the WCET (*Worst Case Execution Time*) of the state transfer to a backup replica.

When the state management of the service is not delegated to the provider, two solutions are possible.

In the first solution (second row in Table 2), the *StateFulReplication* performs checkpointing using state management functions (*Save_State/Restore_State*) developed by the provider. This kind of approach was used in object-oriented fault tolerant systems: an abstract *StateManager* class is provided to the developer who is responsible for the implementation of two virtual methods *Save_State* and *Restore_State* (see. [18]). In our context, these *Save_State* and *Restore_State* operations must be accessible to the ISWD platform to be triggered when appropriate. The WSDL contract can be extended with the signature of these state management operations that are implemented by the provider⁴. Clearly, such operations require authentication to be activated. The IWSD platform can remotely manage the state of WS replicas.

The second solution *LogBasedReplication* relies on session management functions provided by the connector and used by the client (*start_session* and *end_session*). During a session defined by the

⁴ An example of a WSDL contract extended with state management operations can be found at: <http://www.laas.fr/~nsalatge/IWSD>

client (i.e. bracketed by `start_session` and `end_session`), the connector logs all operations in progress. When the primary replica fails, all the saved requests can be resent to a backup replica and replayed to reach a consistent state. This approach enables to undo operations to be performed thanks to information recorded into the logs. The *WS-AtomicTransaction* [19] and *WS-Coordination* [20] mechanisms can be used to this aim.

With active replication strategies, the handling of state management issues can be performed in the same way as before, but is only required for cloning a replica. The most important problems concern the consistency of replicas execution, which essentially depends on inputs delivery and execution determinism of the WS replicas, the later being out of our control.

In both active replication solutions (see row 4-5 on Table 2), the control of requests ordering at the replicate can be done by a connector. However this connector must be the only one able to contact the target WS replicas to ensure requests ordering using *WS-Reliability* [21]. This specification is indeed able to guarantee the delivery, the elimination of duplicates and the ordering of messages. Implementing atomic multicast at the connector level is however complex and costly and certainly need more investigations, because solutions that work “in the small” (group communications providing atomic multicast) do not work “in the large” for obvious performance reasons.

7 Implementation issues and experiments

From an implementation viewpoint, the core software development of this project has been realized with the Xerces-C library. It roughly represents 65000 codes lines corresponding to the implementation of the DeWel compiler and the connector support infrastructure IWSD. The IWSD platform has been installed on a rack of PCs running linux Debian 2.4.

❖ Regarding DeWel, it is clear that the evaluation of a DSL is not always easy as it relates to testing compiler facilities. However, the well-recognized characteristics of a DSL are expressiveness, conciseness and performance, properties that must be enforced as discussed in [22]. The evaluation of expressiveness of a DSL in practice implies the use of a large set of applications. We have used DeWel to produce SFTCs for about 150 WS, and implemented an active replication connector with Amazon⁵. These experiments show that the DeWel expression is 18 times smaller that its counterpart in C++, for a program with empty assertions. We also measured the connector overhead without recovery strategy that is about 3,5% of the response time, which is acceptable for large-scale WS-based applications.

❖ The IWSD platform can be used to perform other measurements. For instance, the availability of various Web Services was evaluated thanks to the error information collected by the *Health Monitor*. The results obtained are given in Figure 7. Although the *Health Monitor* provides other types of information (like session time, numbers of received requests, numbers of communicationException...

⁵ A full account of this example can be found at the url: <http://www.laas.fr/~nalatge/IWSD> (DeWel examples)

etc), we only report here the availability ratio. Approximately 1000 requests were sent to each target service and we observed that the availability of the candidate services could vary a lot during the experiments. For instance, in our experiments, Google has the lowest availability ratio (about 82%) caused by 352 communication errors over 1913 requests sent. Among these errors, 64 are due to a very long response time latency and 288 due to a server unavailability (HTTP Error codes 502: Bad Gateway). Among the 6 replicas of the Amazon WS, only one replica reached an availability ratio close to 100% (i.e. AmazonUK). The failures that occurred were mainly due to communication errors (communicationException), i.e. due to a too long latency. It is worth noting that this depends on a user-defined temporal value representing a specific dependability constraint of the user.

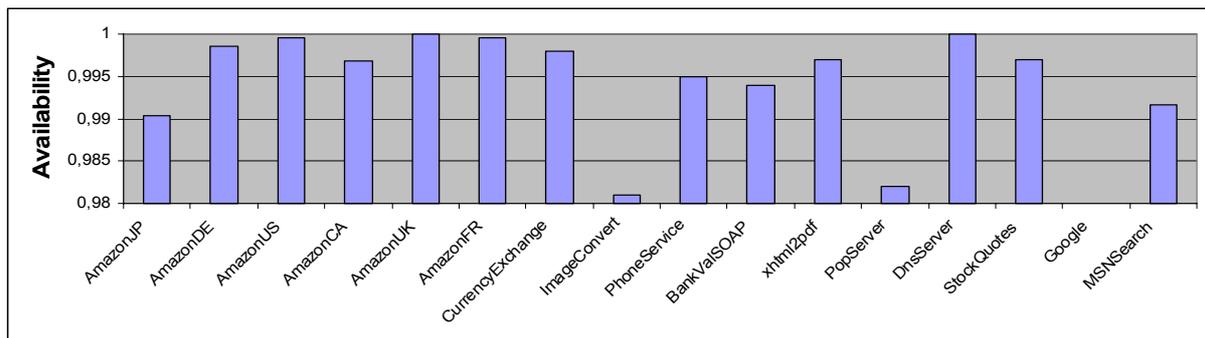


Figure 7: Availability of Web Services

❖ Regarding recovery strategies, we have created a connector selecting the active replication strategy with the various replicates of Amazon. Figure 8 shows the availability results obtained using such recovery strategy. As expected the availability ratio is close to 100%, although the availability of some replicas can be very low, e.g about 70% for AmazonJP. Because most of the faults observed in our experiments were “clean” communication faults, the observed availability was also about 100% with the basic passive replication. During one experiment with the connector using the basic passive replication, we observed that one transient error leading to the reboot of the router has caused the redirection of the request to four replicas in a row before returning a valid response!!!

To measure the overhead of each recovery mechanisms, we have developed connectors using different recovery modes on a simulated StockQuotes WS with three replicas. We compared the overhead with a connector having no recovery mode. The voting replication have the highest overhead (9,66 %). The passive replication has an overhead of 2,18 % whereas the overhead for active replication is of 1,39 %. In the latter case, the response time corresponds to the first replica that transmits a correct response.

❖ Finally, we also performed experiments with equivalent services, through the notion of Abstract Request to a Generic Web Service. Since Google exhibited a weak availability ratio in previous experiments carried out, we have developed a connector able to map requests to both WS of Google and MSN. These services are similar but not identical (different WSDL documents). The results of this experiment are presented in Figure 9. The target Generic Web Service has an availability ratio equal to 100% through this connector. This availability level is reached thanks to the intrinsic availability of

the MSN WS compared to Google WS. Experiments with the basic passive replication strategy show that 6% of the requests have being redirected to the MSN server due to an unavailability of the Google server considered as a primary, this being of course totally transparent to the client.

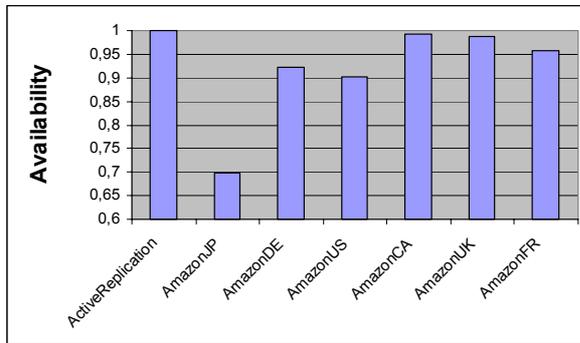


Figure 8: Active Replication with Amazon

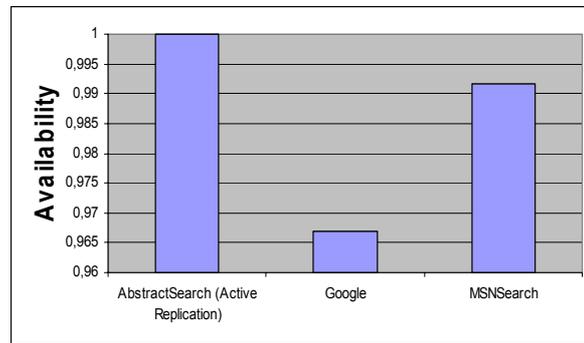


Figure 9: Active Replication with abstract service

8 Related work

Dependability issues are the major topic limiting the deployment of service oriented application in critical domains [23]. The main founders of the Web Services technology (IBM et Microsoft) have spent huge efforts to develop specific WS protocols to guarantee security and safety properties [24]. This specification can be introduced in SOAP parser of the client, of the provider and of the IWSD platform to provide these specific characteristics.

Concerning recovery strategies, the active [25] and passive [26] replication mechanism are already implemented in context of WS by introducing specific middleware respectively named FTWeb [25] and FT-SOAP [26]. These solutions are intrusive and affect the interoperability of WS because they require to install an applicative component on the server side and client side to implement the replication. In both cases, the state management is performed on the server side and the WS replicas must be developed by the same provider. For the active replication with FTWeb, WS-Reliability [21] and [27] are also use to guarantee the determinism of the input requests delivery.

The active replication strategy with vote has also been implemented in Thema [28] and in [29]. In [29], it is achieved on the client side in an ad-hoc way using the “equals” method of Java Objects. This approach strongly relies on the implementation language of the client, which is a restrictive assumption. In Thema, the vote is performed at the server side using a specific library. In our approach, the voting algorithm can be specialized by the users in a DeWel program. The DeWel compiler automatically generates specific objects containing an appropriate equal operator to easily perform the vote in the connector.

The design of a connector for a composite WS (resulting from the aggregation of so-called atomic services, using tools such as BPEL [30], for example) is achieved in the same way. As the implementation of the WS is unknown to the client and so to the IWSD platform, there is no way to distinguish an atomic WS from a composite one. It is however important to mention here that recent

work focused on the reliability of composite Web Services. The idea consists in changing of an atomic service belonging to a composite service by a more recent version [4, 31]. In this work, the provider is in charge of state management issues or a specific active replication is used.

Some related work may help us to perform the evaluation of IWSD. For example, WS-FIT [32], a fault-injection tools for Web Service, can be used to assess the robustness of internal SOAP parser and connector. In a same way, the work realized in [33] inserts mutant in WSDL contract in order to generate mutated Web Service interfaces used to test Web Services. This approach could be used to generate mutated DeWeL templates and to verify the robustness of DeWeL compiler.

9 Conclusion

Although the implementation of WS can be reliable in some respects using conventional techniques, there is no way today to help the client making non-intrusive ad-hoc or customized fault tolerance mechanism for a given usage in a SOA. This is clearly of high interest since client's strategies may change from one SOA to another (resources, networking devices, dependability constraints, evolution requirements, etc.) given that WS can be shared among applications with different dependability constraints. Clearly, guarantees of non-functional properties must be enforced in service contracts, and this work is indirectly a contribution to this aim.

To address this issue and make fault tolerance mechanisms adaptable to clients' needs, we propose an infrastructure enabling clients to develop, manage and execute *Specific Fault Tolerance Connectors* to Web Services. A key feature of this approach is to rely on a Domain Specific Language to develop connectors. DeWeL aims at providing the necessary and sufficient language features to (1) declare and customize built-in fault tolerance strategies and (2) express runtime assertions for each individual operation of a service. The language is supported by a tool suite that applies checks at multiple levels in order to produce robust connectors. In addition, no specific protocol is needed to use DeWeL connectors in practice.

The main benefit of IWSD, the proposed framework and its tools suite, is to enable developers to improve the overall dependability of a Web application realized with existing Web Services, whatever their individual reliability is. Improving the dependability of individual services is of course welcome, this being implementation dependent for stateful Web Services. This framework and tools suite simplifies the adaptation of dependability mechanisms to user's requirements (both clients and providers), because the solution is not intrusive and well integrated into the Web Services world.

The Service Oriented Architecture concept brings the notion of large-scale application to reality, but Internet as a backbone introduces multiple sources of faults by construction. The virtues of this approach are to render applications as dynamic as possible, by picking *Off-The-Net* individual useful services. As a consequence, changes and evolution are core issues of SOA development. This novel situation must be taken into account as far as dependability is concerned, not only for today's

application of these concepts, but also for the future since critical application domains are becoming more and more interested by this approach. We believe that traditional solutions to make individual WS platforms reliable are not sufficient. The language support presented in this paper together with its supporting infrastructure is a contribution to the dependability challenges we are faced today regarding SOA based applications in critical application domains.

10 References

- [1] H. He, "What is Service-Oriented Architecture?," <http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>, Sept, 2003.
- [2] F. Salles, M. R. Moreno, J. C. Fabre, and J. Arlat, "Metakernels and fault containment wrappers," *29th IEEE International Symposium on Fault-Tolerant Computing (FTCS-29), Madison (USA)*, pp. 22-29, 15-18 June 1998.
- [3] D. Cotroneo, C.Di Flora, and S. Russo, "Improving Dependability of Service Oriented Architectures for Pervasive Computing," *In The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*, 2003.
- [4] A. Gorbenko, V. Kharchenko, P. Popov, A. Romanovsky, and A. Boyarchuk, "Development of Dependable Web Services out of Undependable Web Components," *School of Computing Science, University of Newcastle upon Tyne, Technical Report Series CS-TR-863*, october 2004.
- [5] S. Dawson, F. Jahanian, and T. Mitton, "Experiments on six commercial TCP implementations using a software fault injection tool," *Software Practice and Experience*, vol. 27, pp. pp. 1385-1410, Dec. 1997.
- [6] J. Stone and C. Partridge, "When the CRC and TCP checksum disagree," *in Proceedings of the 2000 ACM SIGCOMM Conference*, pp. pp. 309-319, 2000.
- [7] S. S. Yau and R. C. Cheung, "Design of Self-Checking Software," presented at Proceedings of International of Conference on Reliable Software, Los Angeles, CA, USA, IEEE Computer Society Press., 1975.
- [8] J.-C. Laprie, J. Arlat, C. Béounes, and K. Kanoun, "Definition and Analysis of Hardware-and-Software Fault-Tolerant Architectures," *IEEE Computer*, vol. 23, no 7, pp. 39-51, 1990.
- [9] R. Chillarege, "Orthogonal Defect Classification ", E. M. R. L. Handbook of Software Reliability Engineering, McGraw-Hill, Ed., 1995.
- [10] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold, "An Overview of AspectJ " *In Proc. of ECOOP 2001*.
- [11] J. Parra, Sánchez-Alonso, S. S., and O. and Joyanes, "RAWS: Reflective engineering for Web services.," *In Proceedings of ICWS'2004 - IEEE International Conference on Web Services*, pp. 488-497. San Diego, California, USA., 2004.
- [12] M. Debusmann and K. Geihs, "Towards Dependable Web Services," *prdc, 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, pp. 5-14.
- [13] W. C. Recommendation, "XSL Transformations (XSLT)," 16 November 1999.
- [14] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, 2001.
- [15] T. U. Xiang Gao, T. U. Jian Yang, and T. U. Mike. P. Papazoglou, "The Capability Matching of Web Services " *IEEE Fourth International Symposium on Multimedia Software Engineering (MSE'02)* pp. p. 56, 2002.
- [16] S. R. Ponnekanti and A. Fox, "Interoperability among independently evolving web services " *In ACM/Usenix/IFIP Middleware '04, Toronto, Canada*, pp. 331-351, October 2004.

- [17] J. Wu and Z. Wu, "Similarity-based Web Service Matchmaking," *IEEE International Conference on Services Computing (SCC'05)*, vol. Vol-1, pp. pp. 287-294, 2005.
- [18] Graham D Parrington, Santosh K Shrivastava, Stuart M Wheeler, and M. C. Little, "The Design and Implementation of Arjuna," *USENIX Computing Systems Journal*, vol. 8, pp. 253-306, Summer 1995.
- [19] IBM, Microsoft, IONA, BEA, Arjuna, and Hitachi, "Web Services Atomic Transaction (WSAtomicTransaction), Version 1.0," <http://www-128.ibm.com/developerworks/library/specification/ws-tx/#atom>, August 2005.
- [20] IBM, Microsoft, IONA, BEA, Arjuna, and Hitachi, "Web Services Coordination (WSCoordination), Version 1.0," <http://www-128.ibm.com/developerworks/library/specification/ws-tx/#atom>, August 2005.
- [21] SUN, "Web Services Reliable Messaging TC WS-Reliability," <http://www.oasis-open.org/committees/download.php/5155/WS-Reliability-2004-01-26.pdf>, 2003.
- [22] C. Consel and R. Marlet, "Architecturing Software Using A Methodology for Language Development," *In International Symposium on Programming Languages, Implementations, Logics and Programs (PLILP '98), LNCS 1490, Pisa, Italy*, pp. 170-194, 1998.
- [23] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy, "Dependability in the Web Services Architecture," *In Architecting Dependable Systems. LNCS 2677*, June 2003.
- [24] T. S. Donald F. Ferguson, Brad Lovering, John Shewchuk, "Secure, Reliable, Transacted Web Services: Architecture and Composition," *IBM and Microsoft Corporation*, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/wsOverView.asp>, Septembre 2003.
- [25] G. T. Santos, L. C. Lung, and C. Montez, "FTWeb: A Fault Tolerant Infrastructure for Web Services," *In the Proceedings of the 2005 Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'2005) 2005*.
- [26] D. Liang, C.-L. Fang, and C. Chen, "FT-SOAP: A Fault-tolerant web service," *Tenth Asia-Pacific Software Engineering Conference, Chiang Mai, Thailand, 2003*.
- [27] X. Defago, A. Schiper, and P. Urban, "Totally ordered broadcast and multicast algorithms: a comprehensive survey," *Technical Report DSC/2000/036, Dept. of Communication Systems, EPFL, 2000*.
- [28] M. G. A. I. M. Merideth, T.; Tai, S.; Rouvellou, I.; Narasimhan, P, "Thema: Byzantine-fault-tolerant middleware for Web-service applications," *Reliable Distributed Systems, 2005. SRDS 2005. 24th IEEE Symposium* pp. Page(s):131 - 140 2005.
- [29] N. Looker, M. Munro, and J. Xu, "Increasing Web Service Dependability Through Consensus Voting," *the 2nd International Workshop on Quality Assurance and Testing of Web-Based Applications, COMPSAC, Edinburgh, Scotland, July 25-28, 2005*.
- [30] T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. T. (Editor), I. Trickovic, and S. Weerawarana., "Business Process Execution Language for Web Services, Version 1.1," 2003.
- [31] A. Gorbenko, V. Kharchenko, P. Popov, and A. Romanovsky, "Dependable Composite Web Services with Components Upgraded Online " *in Architecting Dependable Systems ADS III, (R. de Lemos, C. Gacek, A.Romanovsky, Eds.)*, vol. LNCS 3549, pp. 96-128.
- [32] N. Looker, M. Munro, and J. Xu, "WS-FIT: A Tool for Dependability Analysis of Web Services," *1st Workshop on Quality Assurance and Testing of Web-Based Applications, COMPSAC, Hong Kong, 28-30 Sep 2004*.
- [33] R. Sibli and N. Mansour, "Testing Web services," *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on 2005*, pp. 135

Improving DBMS Performance through Diverse Redundancy

Vladimir Stankovic, Peter Popov

Centre for Software Reliability,

City University,

Northampton Square,

London EC1V 0HB,

United Kingdom

V.Stankovic@city.ac.uk, Ptp@csr.city.ac.uk

Abstract

Database replication is widely used to improve both fault tolerance and DBMS performance. Non-diverse database replication has a significant limitation - it is effective against crash failures only. Diverse redundancy is an effective mechanism of tolerating a wider range of failures, including many non-crash failures. However it has not been adopted in practice because many see DBMS performance as the main concern.

In this paper we show experimental evidence that diverse redundancy (diverse replication) can bring benefits in terms of DBMS performance, too. We report on experimental results with an optimistic architecture built with two diverse DBMSs under a load derived from TPC-C benchmark, which show that a diverse pair performs faster not only than non-diverse pairs but also than the individual copies of the DBMSs used. This result is important because it shows potential for DBMS performance better than anything achievable with the available off-the-shelf servers.

1. Introduction

The most important non-functional requirements for a Database Management System (DBMS) are performance and dependability, which often require mutually exclusive mechanisms. Thus, a trade-off between the two is sought, which would be optimal for a specific system.

Data replication has proved to be a viable method of enhancing both dependability and performance of DBMSs. Performance is improved by balancing the load between the deployed replicas, while fail-over

mechanisms are normally used to re-distribute the load of a failed replica among the remaining operational ones. Crashes are commonly believed to be the main type of failure of DBMSs. Providing that only crashes occur, using several identical replicas provides appropriate protection. Under this assumption the replication scheme ROWAA (read once write all available) is adequate [1]. Unfortunately, this common belief is hard to justify. In a recent study, we presented overwhelming evidence against crash failures being the main concern [2]. Using the log of known bugs reported for four major DBMSs we observed for all four servers that more than 50% of the known bugs lead to non-crash failures, which will not be tolerated by a non-diverse replication. Only by deploying diverse redundancy, i.e. deploying diverse replicas, would we deliver an adequate protection against the non-crash failures of the DBMSs.

A possible architecture for a fault-tolerant server employing (diverse) redundancy is depicted in

Figure 1. The middleware propagates the statements generated by the client applications to both (all, in case of more than 2) diverse replicas for execution. The results from the replicas are collected by the middleware and in the case of a positive adjudication the middleware reports a result back to the client application(s). Clearly, this architecture differs from the ROWAA scheme. In the new architecture all statements (including the reads from the database) are executed multiple times by several diverse replicas, while in the ROWAA scheme all active replicas execute only the writes to the databases.

While dependability gains from deploying diverse redundancy are beyond doubt, it is far from obvious what the implications of this architecture would be for system performance. From the known applications of design diversity in other areas, it is well known that

© 2006 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

fault-tolerant mechanisms (failure detection, fault-containment, state recovery, etc.) have their performance cost. Is diverse redundancy then necessarily a bad thing in terms of system performance?

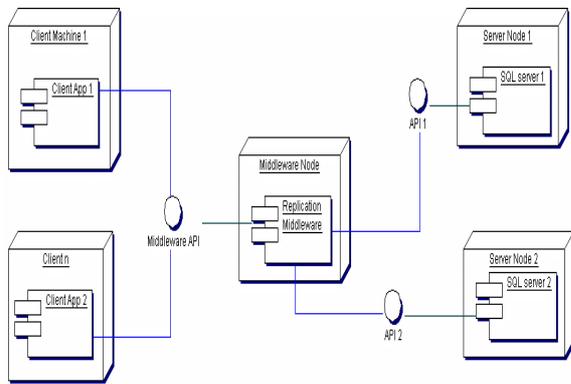


Figure 1. Fault-tolerant server node (FT-node) with two (possibly more) diverse DBMSs (SQL server 1 and SQL server 2). The middleware “hides” the servers from the clients (1 to n) for which the data storage appears as a single DBMS.

The overall performance of the system shown in

Figure 1 will depend on the performance of the diverse replicas deployed and on the performance characteristics of the middleware itself. For instance, the middleware can use different adjudication mechanisms. A few reasonable alternatives are listed below:

- *Slowest response.* The middleware collects the results of the *individual statements* (a multitude of which constitute a whole transaction) executed by diverse replicas. Once a sufficient number of responses are collected, they are adjudicated and only if identical responses from all the replicas are observed a successful completion of the statement is reported back to the client application.
- *Fastest response.* Alternatively, the middleware may buffer the statements coming from a client application and make them available to the diverse replicas as soon as the statements are placed in the respective buffers. Each diverse replica collects the next available statement from its respective buffer, executes it, marks it as being completed and makes the response from the statement available to the middleware. As soon as the

middleware receives the first response to a statement from a replica, it is immediately passed on to the client application, thus letting the client application proceed with the other statements within the transaction. The fastest response comes from either of the DBMSs, depending on the SQL statement (**Figure 2**). Responses from the diverse replicas to the same statement are adjudicated later, when a sufficient number of responses are collected, but before the end of the transaction. Buffering the statements in the middleware allows the diverse replicas to work at a maximum speed within transactions, as shown in **Figure 2** (DBMS1 would start execution of the next SQL statement even though the DBMS2 has not finished the previous one as indicated with the dashed rectangle). The transactions are committed (or aborted) based on the outcome of adjudicating the results of the statements. Commit is only applied if all the replicas execute all the statements successfully and all the statement responses are positively adjudicated. Otherwise, the transaction is aborted.

- *Optimistic response.* This is similar to the fastest response except: i) *no adjudication* of the responses from the diverse replicas is applied; ii) a *skip* feature is implemented in the middleware as follows. Before a replica, X, executes a *read* (i.e. SELECT) *statement* it checks if a response to this statement has already been received from another replica, $Y \neq X$. If so then X does not execute the statement (i.e. skips it)¹. The modifying SQL statements (DELETE, INSERT and UPDATE) are executed on all servers, i.e. they cannot be skipped. Clearly, this regime of operation does not offer the same level of protection as the previous ones. It may, however, be adequate in many cases, which we discuss later (see the Discussion section).

We have already [3] on systematic differences between the times it takes diverse DBMSs to execute the same statement. This may be due, for example, to the respective execution plans being different, the concurrency control mechanisms being implemented differently, etc. When the *slowest response* regime is used such differences will lead to the fault-tolerant

¹ The functionality of looking up the next statement and the ‘skip’ feature is, of course, implemented in the middleware, which relays to the DBMSs the statements for execution. If a read statement is to be skipped, then the middleware simply does not pass it to the respective DBMS for execution.

node (FT-node) being slower than the respective DBMSs it consists of. When the *optimistic regime* is used, however, the systematic difference might lead to improved performance. If the mix of statements within a transaction is such that both servers ‘skip’ statements, then the transaction will take the FT-node shorter than it would take each of the DBMSs it uses. When the ‘skip’ feature is not used the best that the FT-node can do is process SQL statements as fast as the faster of the two servers can, thus diversity cannot bring any performance gains.

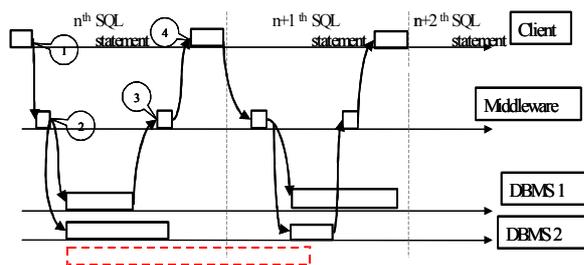


Figure 2. Timing diagram of a client communicating with two, possibly diverse, database servers and the middleware running in fastest response or optimistic regime. The meanings of the callouts are: 1 – the client sends an SQL statement to the middleware, 2 – the middleware translates the request to the dialects of the servers and places the resulting SQL statements, or sequences of SQL statements, in the respective server buffers; 3 – the fastest response is received by the middleware; 4 – the middleware sends the response to the client. The dashed rectangle indicates that DBMS2 will not be ready to start (n+1)th SQL statement at the same time with DBMS1

This paper, therefore, is focused on the empirical investigation of whether the potential performance gains with the optimistic regime of operation of the FT-node can be achieved under a realistic load, such as the one defined by the TPC-C performance benchmark [4]. Whichever regime under the FT-node operates, data consistency between the diverse replicas must be guaranteed, which is typically defined as *1-copy serialisability* between the transaction histories of the replicas [1].

Should a level of replication be required that is higher than the number of diverse replicas used in a

single FT-node, then the FT-node can be combined with any database replication scheme, which is considered adequate for a particular set of requirements. These can be schemes for *eager* database replication, e.g. based on group communication primitives [5], or even *lazy* replication [6]. In either case the FT-node will replace a replica of a particular DBMS used by the particular database replication scheme.

This paper is structured as follows. In section 2 we describe the experimental setup. In section 3 we enumerate possible configurations of the FT-node. In section 4 we show consistency of the experimental results. In section 5 we present the performance comparison of different FT-node configurations. In section 6 we compare performance of the diverse pair and a non-diverse solution. In section 7 we discuss possible performance gains when diverse DBMSs are used and in section 8 we present conclusions made and describe provisions for future work.

2. Experimental Setup

In the empirical study we used our own implementation of the industry-standard benchmark for online transaction processing - TPC-C [4], to evaluate the potential for performance improvement. TPC-C defines five types of transactions: *New-Order (NO)*, *Payment (P)*, *Order-Status (OS)*, *Delivery (D)* and *Stock-Level (SL)* and sets the probability of execution of each. The minimum probability of execution for each transaction type is as follows: NO – 43%, P – 43%, OS – 4%, D – 4% and SL – 4%. The benchmark provides a mechanism for performance comparison of the DBMSs from different vendors, with different hardware configurations and operating systems. The specified measure of throughput is the number of *NO* transactions completed per minute (under the specified mix of transaction types). Our measurements were more detailed than those required by the standard. We recorded the response times of the individual SQL statements and transactions executed by the DBMSs used in the FT-node. The test harness consisted of three machines:

- a client machine, which executes a JAVA implementation of the TPC-C standard (it uses JDBC to access the DBMSs);
- two server machines, on which two diverse open-source DBMSs are run, namely InterBase 6.0 and PostgreSQL 7.4.0 (referred to as IB and PG, respectively, in the rest of the paper).

The two DBMSs ran on Linux RedHat 6.0 (Hedwig) operating system, while the client machine ran under Windows 2000 Professional (sp4) operating system. The hardware specifications are as follows:

- *client machine*: 1.5 GHz Intel Pentium 4 processor, 640 MB RAMBUS RAM and 20GB HDD (Maxtor DiamondM)
- *server machines*: 1.5 GHz Intel Pentium 4 processor, 384 MB RAMBUS RAM and 20GB HDD (Seagate U Series).

The implementation of the TPC-C application did not necessitate the use of any proprietary features from either IB or PG. The SQL statements were implemented using the common subset of the language. Nevertheless we have developed preliminary versions of our own SQL translator tool. In addition one could make use of commercial products for porting between different DBMSs such as *Fyrcle* [7], Oracle-mode Firebird or its PostgreSQL counterpart *EnterpriseDB* [8].

3. FT-node Configurations

We run a set of experiments with the following server configurations:

- 1IB1PG, an FT-node with a copy of IB and PG. 1IB,
- a single replica of IB;
- 1PG, a single replica of PG;
- 2IB, an FT-node with two replicas of IB, and
- 2PG, an FT-node with two replicas of PG.

Each experiment comprises the *same sequence* of 10,000 transactions and was repeated five times, for reasons detailed below. The server machines were restarted and databases restored between the repetitions.

All the measurements were associated with a single TPC-C client under different *server loads* as follows:

- no additional clients;
- 10 additional clients, and
- 50 additional clients.

Whenever additional clients were deployed they executed a mix of read-only transactions (RO mix) instead of the mix of transactions recommended by the TPC-C². The RO mix consists of the two read-only

² We did run multiple concurrent TPC-C clients with our own implementation of 1-copy serialisability between the DBMSs. These experiments, however, did lead to a very large number of non-serialisable transactions, which had to be aborted. Due to non-determinism between the orders in which the servers serve the concurrent clients we could not achieve a repeatable set of

transactions: *Order-Status* and *Stock-Level* of almost equal proportion. Thus, only one TPC-C compliant client modifies the database. The *readers* and *writers* do not conflict in the two DBMSs, since both IB and PG implement a type of MVCC (Multi-Version Concurrency Control). Hence data consistency between the replicas is guaranteed (experimentally confirmed by successfully running a comparison between the databases at the end of the experiments).

The overhead that the test harness introduces (mainly due to using JAVA multi-threading for communication of the clients with the middleware and of the middleware with the different DBMSs) is the same irrespective whether a single or two replicas are used in the experiment. It has been measured to be negligible compared with the time taken by the respective DBMSs to process the 10,000 transactions.

4. Confidence in the Results

Each experimental setup (with a fixed configuration and load) was repeated *five times* so that we could detect significant variation between the observed results due to factors beyond our control (e.g. fragmentation of files on the servers).

Figure 3 shows the mean transaction times for all transactions together and per transaction type in a 10,000-transaction run, grouped by experiment repetitions when only a single TPC-C compliant client is deployed. There is no significant variation between the results across the repetitions. This is true for both a particular transaction type and all transactions together.

experiments to make a fair comparison between the different server configurations. Thus, we chose to restrict the results presented here to experiments with a single TPC-C compliant client while simulating the increased load by deploying an increasing number of read only clients.

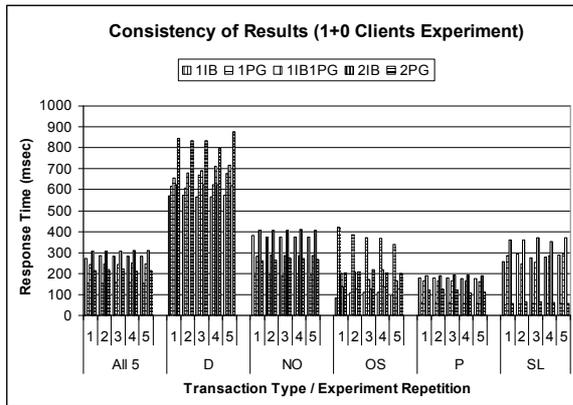


Figure 3. Mean transaction times per transaction type and for all transactions together for 5 repetitions with each of the configurations (1IB, 1PG, 1IB1PG, 2IB, 2PG) with a load generated by a single TPC-C compliant client.

A similar picture, consistent across the repetitions, was established for the increased load of 10 and 50 additional clients (Figure 4). The only configuration with a noticeable variation between the repetitions was 1IB. In particular, the first run is 25% faster than the remaining four in terms of the mean transaction time with all transaction types (represented by the first bar in each of the five groups above the “All 5” category). A noticeable variation also exists between the specific transaction types, for which the percentages vary between 20% and 25%. This variation, however, does not change the ordering between the configurations.

In addition the ordering between the configurations does not change even if we execute a different sequence of transactions. This was experimentally confirmed by executing 10,000 transactions in different order with either a single TPC-C compliant client or with ten additional clients.

Such consistency between the observations, in particular, the fact that the ordering between the configurations remains unchanged across the repeated experiments, is the reason why in the rest of the paper we compare the performances using a single run per configuration.

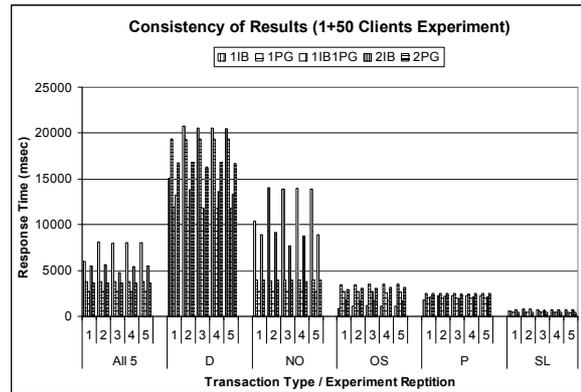


Figure 4. Mean transaction times per transaction type and for all transactions together for 5 repetitions of each experiment type (1IB, 1PG, 1IB1PG, 2IB, 2PG) under increased load with 50 additional read-only clients.

5. Performance Comparison of Different DBMS Configurations

To compare different DBMS configurations we used the following measures of interest:

- mean transaction time (for all five transaction types);
- mean transaction time for a particular type of transaction;
- cumulative transaction time, i.e. experiment duration.

Figure 5 depicts the response time when only a single TPC-C client communicates with the FT-node configurations. 1PG is on average the best configuration under this load, though transactions of type Delivery and Order-Status are faster on 1IB. The ranking changes when the load increases (Figure 6). Now the fastest configuration on average is the diverse pair, albeit not for all transaction types (1IB is the fastest for Order-Status and Payment, while 1PG is the fastest for Stock-Level). The figure indicates that the diverse DBMSs “complement” each other in the sense that when IB is slow to process a transaction then PG is fast (New-Order and Stock-Level) and vice versa (Payment, Order-Status and Delivery). These systematic differences illustrate why the 1IB1PG diverse pair is the best configuration on average. In addition the ‘skip’ feature enables the diverse pair to

augment this advantage by omitting the read (SELECT) SQL statements on the slower DBMS.

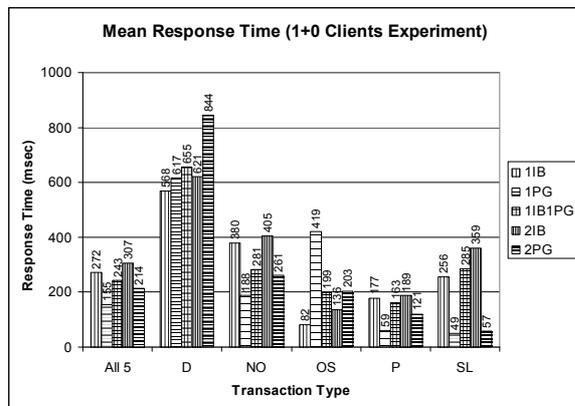


Figure 5. The mean transaction times for each transaction type and for all transactions together under a load generated by a single TPC-C compliant client. The configurations compared under this load are as follows: configurations with a single DBMS (1IB, 1PG), a configuration with a diverse pair of DBMSs (1IB1PG) and configurations with homogenous pairs of DBMSs (2IB, 2PG).

Although a DBMS is fastest on average for a particular transaction type, within the transactions the fastest responses to SQL statements may come from different DBMSs. This fact is utilised in the diverse pair. Hence, it is not surprising that IB executes more SELECT statements in an experiment than PG when the two are employed as a diverse pair (IB executes 70%, while PG executes 51%)³.

Similar results were obtained under the load with 50 additional clients.

Figure 7 shows how the ordering changes between the configurations as a result of a load increase. An experiment comprising 10,000 transactions under the 'lightest' load (0 additional clients) is fastest with 1PG. Under increased load, however, the diverse pair, 1IB1PG, becomes the fastest configuration. The experiment duration with the diverse pair is shorter than with the individual DBMSs, or with either of the

³ There is nothing unusual in the fact that the sum 70% + 50% is greater than 100%. It simply means that there are statements which are executed by both servers. If the fastest server has not completed a statement by the time the slower is ready to start, then both will process the particular statement.

non-diverse (homogenous) DBMS pairs. The diverse pair is 20% faster than the second best configuration (1PG) with 10 additional clients and more than 25% faster than the second best combination (2PG) with 50 additional clients. The benefits of the systematic difference in transaction times between the diverse DBMSs and the efficiency of the 'skip' feature become more clearly pronounced when the load increases.

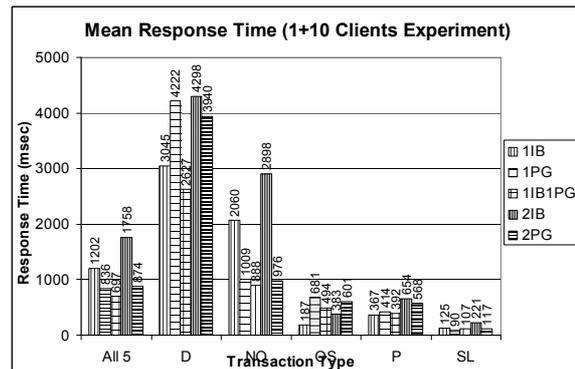


Figure 6. The mean transaction times for single DBMS configurations (1IB, 1PG), diverse DBMSs pair (1IB1PG) and homogenous DBMS pairs (2IB, 2PG) for each transaction type and for all transactions together under an increased load with 10 additional read-only clients.

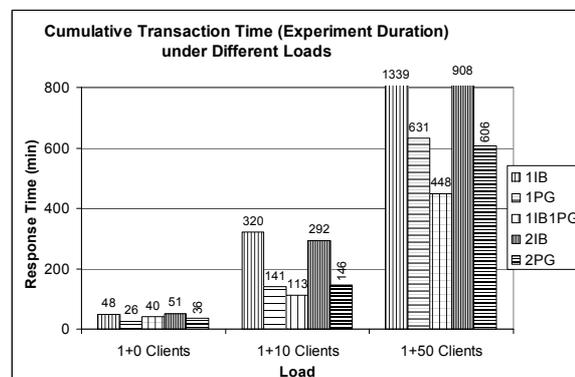


Figure 7. Cumulative transaction time (experiment duration) for the five DBMS configurations under different load (0, 10 and 50 additional read-only clients).

6. Comparison of Diverse Pair and a Non-Diverse Solution

In this section we compare the performance of the diverse pair and of a well-known solution for eager data replication [9]. The solution uses non-diverse redundancy. It combines transactional concurrency control and group communication primitives in order to guarantee data consistency (referred to as TCC+GCP in the remainder of the document) among replicas. It provides both fault-tolerance and good performance.

In order to guarantee a 1-copy serialisability, TCC+GCP relies on “totally ordered” [9] delivery of transactions using a reliable multicast protocol. It is based on the ROWAA (Read-Once Write All Available) protocol [1].

Under this replication scheme the clients served by TCC+GCP connect to only one replica, called the *local replica* of the client. For this client the other replicas of the TCC+GCP are remote replicas. A read-only transaction generated by a client is executed by the local replica, only. A *write* transaction (i.e. one that includes write statements) is first executed by the local replica. The outcomes of the write statements are then broadcast (by the respective middleware) to the remote replicas of TCC+GCP in the form of *write sets*. The remote replicas install the *write sets* according to the total order of transactions established among the replicas used in TCC+GCP.

Clearly, with TCC+GCP the read-only transactions are load-balanced between the replicas. Ideally, the clients should be fairly divided between the replicas. A fair performance comparison, thus, of TCC+GCP and an FT-node with two diverse DBMSs, would require the following arrangement:

- a single DBMS working in TCC+GCP will be subjected to the write transactions load generated by all clients and half of the load generated by the read-only transactions generated by the clients;
- the FT-node handles the entire load, both from write and read transactions, generated by all the clients.

We ran experiments for loads generated by a single TPC-C client and additional read-only clients: 10 and 50. To make a fair comparison between an FT-node and TCC+GCP, we used the results measured for the FT-node (see above) and run a new set of experiments with 5 and 25 read-only clients respectively with an FT-node. We *simulated* the performance of the TCC+GCP using the measurements obtained with the

FT-node under the new loads (with 5 and 25 read-only clients).

We calculated a *lower* and an *upper bound* on the TCC+GCP transaction times as follows. The lower bound is the *actual transaction time* measured in the new experiments with the individual DBMSs and the number of read-only clients equal to 5 and 25, respectively. This lower bound seems unattainable by TCC+GCP, because the installation of the *write sets* (especially the *lock phase*) [10] on the remote replicas, as well as on the local replicas is not accounted for in the lower bound. Installing the write sets is on the *critical path* – it is always done *after* the local replica creates the write sets. The *upper bound is calculated* from the experimental log (in which we record the start and completion times of the individual statements) by doubling the execution time of all *write* SQL statements (DELETE, INSERT and UPDATE) encountered during the experiment. Whether the bound is indeed an upper bound is moot since it is unclear whether the actual overhead due to group communication primitives and the actual installation of the *write sets* by all the replicas is greater or smaller than the time it takes the local replica to execute a write statement. The upper bound may be too pessimistic, if the mentioned overheads are negligible compared with the write statements execution times. On the other hand, however, a simplistic implementation of the write sets would be forwarding them to the remote replicas, which in turn will actually execute them. Under this simplistic scenario, our upper bound will be in fact too optimistic because it does not account for the overhead due to propagating the write sets to the remote replicas. In summary, the realism of the upper bound is questionable and should be scrutinised in the future, ideally by actually implementing TCC+GCP. Despite this problem, however, using the lower and the upper bounds allows us to get preliminary indications of how the performance of FT-node compares with TCC+GCP.

Figure 8 presents the results of a fair comparison between the two replication schemes under a load with 1 modifying and 50 additional read-only clients. Diverse pair performs clearly better than TCC+GCP if IB is used: the transaction times of the FT-node is lower than the lower bound (unattainable by TCC+GCP). The diverse pair is also better (~ 15%) than the upper bound of TCC+GCP with PG. It is, however, worse than the lower bound of TCC+GCP with PG. The diverse pair is ~30% slower than the lower bound. Thus, it remains unclear whether

TCC+GCP with PG is faster than the FT-node. Similar results have been observed in all repetitions with this load.

Similar ordering between the FT-node and the TCC+GCP has been observed with a lower load of only 10 additional read-only clients. Again, only the lower bound of the non-diverse replication with PG is faster than the diverse pair. However, the difference between the diverse pair and the upper bound is smaller.

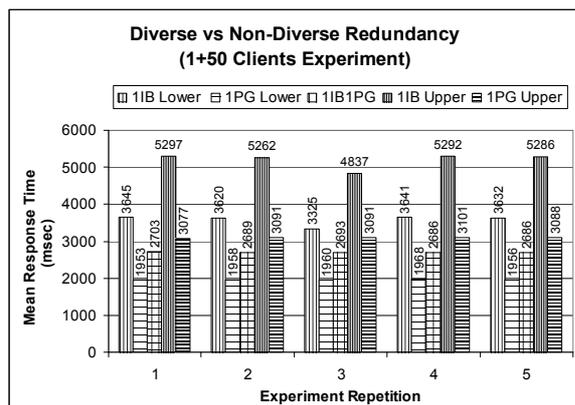


Figure 8. Mean response times for diverse replication (1IB1PG) and calculated lower and upper bound of mean response times for a non-diverse replication schema when either 1IB or 1PG is used. The lower bound is calculated using results from respective individual DBMS (IB or PG) experiment under the load with 1 modifying and 25 read-only clients. The number of read-only clients was halved because the non-diverse schema uses a load balancing approach where reads are executed at only one node. To estimate the upper bound, mean response time of transactions' write sets was added to the lower bound.

7. Discussion

Performance implications of using diverse redundancy in the context of database replication are the focus of this paper. Diverse redundancy is the only known realistic protection against *design faults* in complex software products. Once diverse redundancy is deployed there might exist performance implications, which we evaluated empirically.

A standard fault-tolerant architecture (see Figure 1) would dictate adjudicating all the responses from (a sufficient number of) diverse replicas before a response is returned back to a client application, i.e. adjudication is applied at the level of individual statements. This adjudication, normally implemented by a specialised middleware, can be done as the responses are received (referred to as the *slowest response*) or postponed and completed before the end of the corresponding transaction (the *fastest response* regime). Either way, fault-tolerance will lead to performance penalty and the FT-node cannot be faster than the fastest of the deployed DBMSs.

The schemes adopted for practical database replications provide no protection against design faults. A common assumption is made that node crashes are the main concern, an assumption under which various optimistic regimes of operations are used such as ROWAA. These do not require statement adjudication and as a result the adjudication overhead is simply avoided.

Failures of DBMSs are rare. Most of the time the applications use statements that are handled correctly by the deployed DBMSs. Even if diverse DBMSs are deployed most of the statements will be handled correctly by all the diverse replicas. Thus, most of the time adjudicating the responses of diverse replicas will reveal no discrepancy, making the adjudication overhead a waste of time. The point, of course, is that we will never know which statement will turn out to trigger a fault in the DBMSs and revealing a discrepancy between the replica responses. In some extreme cases, however, *one may know with certainty*, that all the statements used by the application will be processed by the DBMSs correctly; hence one may be prepared to use regimes in which the adjudication is eliminated. One such example is the implementation of the optimistic regime. Its advantage compared with the well-known ROWAA regime of operation lies in the fact that under ROWAA the load is statically distributed between the replicas – in the ideal case a fair load-balancing between the replicas is sought. Instead, when the FT-node operates under the optimistic regime its diverse replicas *naturally get the load that they are better at executing*. As a result the optimistic mode has the potential of performing better than ROWAA. Unfortunately, our experiments did not provide a conclusive answer as to whether this potential can be materialised, but it did not refute it either. Further, more accurate measurements, possibly

using proper implementation of the replication schemes based on ROWAA will provide a definitive answer.

It is worth pointing out that the 3 regimes of operation of the FT-node listed above (slowest response, fastest response and optimistic) are not mutually exclusive. In fact, they can be combined to offer *configurable quality of service*, as we argued elsewhere [3]. The clients mainly concerned with high dependability assurance can be served under the slowest response regime. The clients mainly interested in maximising the performance can be served under the optimistic regime of operation. Finally, by deploying learning capabilities, e.g. based on Bayesian inference, [11], the middleware may become capable of switching intelligently between the different regimes of operation: initially a new type of statement (e.g. SQL statement involving a complex and rarely executed JOIN operation) will be treated by the middleware with suspicion and the most conservative, slowest response, regime of adjudication will be applied. As more instances of the same statement are executed successfully (i.e. the adjudication is passed successfully in all the observed instances), then the middleware will switch from slowest response through the fastest response eventually to the optimistic regime of operation. Clearly, adjudication is simply impossible with ROWAA, thus the scope for trading-off intelligently performance for improved dependability assurance is very limited, if not impossible.

8. Conclusions and Future Work

The results presented here show an intriguing possibility to get a *performance gain*, in some cases very substantial, when diverse redundancy is used in the context of database replication. We compared diverse with non-diverse redundancy using an optimistic architecture, FT-node, in which the variation between the execution times of the diverse replicas is turned into a performance advantage. In this setup, diverse redundancy is clearly beneficial compared with non-diverse redundancy.

We also compared non-replicated solutions (a single copy of a DBMS) with an FT-node, in which a diverse pair of DBMSs is deployed. Diverse pair performs significantly faster than the non-replicated solution.

These two results seem very significant since they open up new ways of achieving high performance, especially when the main system concern is achieving as high a performance as possible.

We also looked at how diversity performs against eager replication solutions based on total transaction order (TCC+GCP), which use load balancing for improved performance. The comparison, performed under various simplifying assumptions, is *indecisive* in the general case. Diverse redundancy is not guaranteed to always achieve a known lower bound of performance for those solutions, although we recorded that the diverse pair performed better than TCC+GCP implemented with replicas of Interbase 6.0. This lower bound, however, is unattainable for TCC+GCP too! The performance of diverse redundancy is better than the likely upper bound on the performance of TCC+GCP with replicas of PostgreSQL 7.4. In some cases of simple implementations of TCC+GCP, e.g. the write sets being propagated to the remote replicas in the form of full SQL statements, the upper bound will become a lower bound on the performance of TCC+GCP. For this implementation of TCC+GCP we have a decisive argument in favour of diverse redundancy: it is guaranteed to be faster than TCC+GCP.

In the experiments we used a synthetic load (TPC-C) mainly due to the wide acceptance of the benchmark for performance measurement studies. Although the reported effect is dependant on the mix of SQL statements used we expect similar results in favour of diverse redundancy to be observed for a wide range of real loads. In fact, TPC-C specifies a write intensive mix of statements, not ideal for the optimistic regime of an FT-node. Applications based towards read-only mixes of SQL statements are more suitable for the reported effect to make a bigger impact.

A promising direction for future development is implementation of a configurable middleware, deployable on diverse DBMSs, which would allow the clients to request *quality of service* in line with their specific requirements for performance and dependability assurance.

Acknowledgements

This work has been supported in part by the “Interdisciplinary Research Collaboration in Dependability” (DIRC) project funded by the U.K. Engineering and Physical Sciences Research Council (EPSRC).

Bibliography

1. Bernstein, A., V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. 1987, Reading, Mass.: Addison-Wesley.
2. Gashi, I., P. Popov, and L. Strigini. *Fault diversity among off-the-shelf SQL database servers*, in *International Conference on Dependable Systems and Networks*. 2004. Florence, Italy: IEEE Computer Society Press.
3. Gashi, I., et al., *On Designing Dependable Services with Diverse Off-The-Shelf SQL Servers*, in *Architecting Dependable Systems II*, R. de Lemos, C. Gacek, and A. Romanovsky, Editors. 2004, Springer. p. 191-214.
4. TPC, *TPC Benchmark C, Standard Specification, Version 5.0*. 2002, Transaction Processing Performance Consortium.
5. Patino-Martinez, M., et al. *Scalable Replication in Database Clusters*. In *International Conference on Distributed Computing, DISC'00*. 2000: Springer.
6. Gray, J. and R. Andreas, *Transaction processing: concepts and techniques*. 1993: Morgan Kaufmann.
7. Fyrracle,
http://www.janus-software.com/fb_fyrracle.html.
2006.
8. EnterpriseDB,
http://www.enterprisedb.com/products/migration_toolkit.do. 2006.
9. Jimenez-Peris, R., M. Patino-Martinez, and G. Alonso. *An Algorithm for Non-Intrusive, Parallel Recovery of Replicated Data and its Correctness*. In *21st IEEE Int. Conf. on Reliable Distributed Systems (SRDS 2002)*. 2002. Osaka, Japan.
10. Kemme, B., A. Bartoli, and O. Babaoglu. *Online Reconfiguration in Replicated Databases Based on Group Communication*. In *Int. Conf. on Dependable Systems and Networks (DSN 2001)*. 2001. Goteborg, Sweden: IEEE.
11. Gorbenko, A., et al., *Dependable Composite Web Services with Components Upgraded Online*, in *Architecting Dependable Systems ADS III*, R. de Lemos, C. Gacek, and A. Romanovsky, Editors. in print, Springer. p. 96-128.

Intrusion-Tolerant Middleware

The Road to Automatic Security

The pervasive interconnection of systems throughout the world has given computer services a significant socioeconomic value that both accidental faults and malicious activity can affect. The classical approach to security has mostly consisted of trying to prevent bad things

that can be introduced during the system's development or operation. For example, as a step in an overall plan of attack, an attacker might introduce vulnerabilities in the form of malware.

Attacks are malicious faults that attempt to exploit one or more vulnerabilities. An attack that successfully exploits a vulnerability results in an intrusion, which is normally characterized by an erroneous system state (for example, a system file with unwarranted access permissions for the attacker). If nothing is done to handle these errors, a security failure will probably occur. Attacks often assume the form of inconsistent interactions with different legitimate participants in order to confuse them. Resilient systems should be able to handle these so-called *Byzantine faults*.

Figure 1a represents a fundamental sequence: attack → vulnerability → intrusion → error → failure. This well-defined relationship is called the *AVI/fault model*.

Classical security methodologies mainly focus—quite successfully—on preventing intrusion. However, as reality painfully proves every day, it's impossible, even infeasible, to guarantee perfect prevention: simply put, we can't handle all attacks because they aren't all known, and new ones appear constantly. As a consequence, a few inconspicuous weaknesses are easy prey to hackers, and what's worse, the resulting intrusions that escape the *intrusion-prevention* barrier, as Figure 1b suggests, will go unnoticed and will likely cause security failures.

The last resort is intrusion tolerance, which, as the name suggests, acts

from happening—by developing systems without vulnerabilities, for example, or by detecting attacks and intrusions and deploying ad hoc countermeasures before any part of the system is damaged. But what if we could address both faults and attacks in a seamless manner, through a common approach to security and dependability? This is the proposal of *intrusion tolerance*, which assumes that

- systems remain somewhat faulty or vulnerable;
- attacks on components will sometimes be successful; and
- automatic mechanisms ensure that the overall system nevertheless remains secure and operational.

No large-scale computer network can be completely protected from attacks or intrusions. Just as chains break at their weakest link, any inconspicuous vulnerability left behind by firewall protection or any subtle attack that goes unnoticed by intrusion detection will be enough to let a hacker defeat a seemingly powerful defense. Using ideas from fault tolerance that put emphasis on automatically detecting, containing, and recovering from attacks, the European project MAFTIA (Malicious-and Accidental-Fault Tolerance for Internet

Applications; www.maftia.org) set out to develop an architecture and a comprehensive set of mechanisms and protocols for tolerating both accidental faults and malicious attacks in complex systems. Here, we report some of the advances made by the several teams involved in this project, which brought together international expertise in the areas of information security and fault tolerance.

Intrusion tolerance in a nutshell

Building an intrusion-tolerant system to arrive at some notion of intrusion-tolerant middleware for application support presents multiple challenges. Surprising as it might seem, intrusion tolerance isn't just another instantiation of accidental fault tolerance.

To capture the essence of intrusion tolerance, we must first consider that an intrusion is in fact a malicious fault that has two underlying causes: a weakness, flaw, or *vulnerability*, or a malicious act or *attack* that attempts to exploit the former.

Attacks, vulnerabilities, and intrusions

Vulnerabilities are the primitive faults within a system—in particular, design or configuration faults—

PAULO E. VERÍSSIMO AND NUNO F. NEVES
University of Lisbon, Portugal

CHRISTIAN CACHIN AND JONATHAN PORITZ
IBM Zurich Research

DAVID POWELL AND YVES DESWARTE
Laboratory for Analysis and Architecture of Systems, CNRS

ROBERT STROUD AND IAN WELCH
University of Newcastle upon Tyne

Related work in intrusion tolerance

One of the first architectural attempts to build a secure and robust architecture was Delta-4, a system developed in a European project in the 1980s.¹ It provided distributed intrusion-tolerant services for data storage, authentication, and authorization.

More recently, several projects have also addressed this problem, providing secure middleware,² secure group or broadcast communication,³ or secure authentication.⁴ Oasis is a large US program on intrusion-tolerance research comparable to MAFTIA.⁵

References

1. D. Powell et al., "The Delta-4 Approach to Dependability in Open Distributed Computing Systems," *Proc. 18th IEEE Int'l Symp. Fault-Tolerant Computing*, IEEE CS Press, 1988, pp. 246–251.
2. M.K. Reiter, "Distributing Trust with the Rampart Toolkit," *Comm. ACM*, vol. 39, no. 4, 1996, pp. 71–74.
3. Y. Amir et al., "Secure Group Communication in Asynchronous Networks with Failures: Integration and Experiments," *Proc. 20th IEEE Int'l Conf. Distributed Computing Systems*, IEEE CS Press, 2000, pp. 330–343.
4. L. Zhou, F. Schneider, and R. van Renesse, "COCA: A Secure Distributed On-Line Certification Authority," *ACM Trans. Computer Systems*, vol. 20, no. 4, 2002, pp. 329–368.
5. J.H. Lala, ed., *Foundations of Intrusion Tolerant Systems*, IEEE CS Press, 2003.

after intrusion and before failure. Intrusion-tolerance techniques are in essence automatic, relying on local mechanisms and distributed protocols, and assume combinations of detection (of corrupted hosts or tampered communications), recovery (neutralization of intruder activity), or masking (use of spare components or replicas, such that the whole resists the intrusion of a minority).

Trust and trustworthiness

The relationship between the notions of *trust* and *trustworthiness* is important to understand how intrusion tolerance can lead to secure designs.

Let's consider trust to be a component's accepted dependence on a set of (desirable) properties of another component, subsystem, or system.¹ If *A* trusts *B*, then *A* accepts that a violation of *B*'s properties might compromise *A*'s correct operation. It might also happen that those properties *A* trusts don't correspond quantitatively or qualitatively to *B*'s actual properties. Thus trustworthiness is the measure in which a component (say, *B*) meets a set of properties. Clearly, a robust design implies that trust in *B* should be placed to the extent of *B*'s trustworthiness—that is, the relation "*A* trusts *B*" should imply *A*'s substantiated belief that *B* is trustworthy in the measure of *B*'s trustworthiness.

There is a separation of concerns between how to make a component

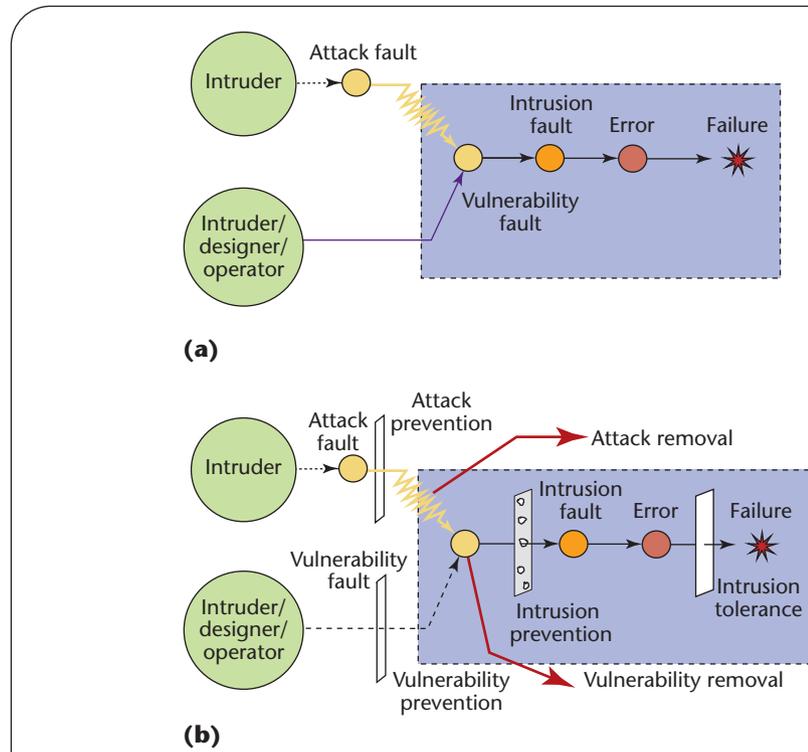


Figure 1. Intrusion sequence. In the (a) attack-vulnerability-intrusion (AVI) fault model, an attack hits a vulnerability, causing an intrusion which, if not handled, will cause a failure; (b) intrusion tolerance, the last resort for protection.

trustworthy (constructing the component) and what to do with the trust placed in the component (building fault-tolerant algorithms). These iterative chains of trust-trustworthiness relations, with the proper specification and verification tools—lead to very clear arguments about system security and dependability.^{2,3}

MAFTIA architecture

The MAFTIA architecture selectively uses intrusion-tolerance mechanisms to build layers of progressively more trusted components and middleware subsystems from baseline untrusted components (hosts and networks). This leads to an automation of the

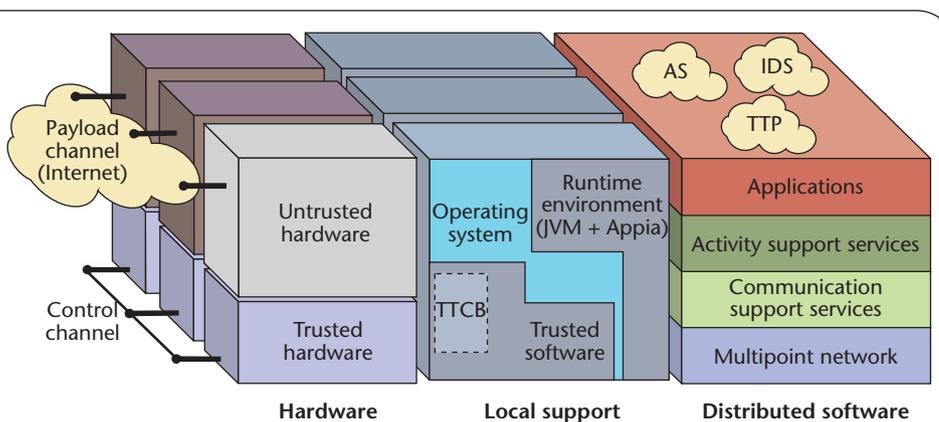


Figure 2. The MAFTIA architecture's three dimensions. Hardware, local support, and distributed software help applications operate securely across several hosts, even in the presence of malicious faults.

process of building resilience: at lower layers, a trustworthy communication subsystem is constructed with basic intrusion-tolerance mechanisms. Higher-layer distributed software can then trust this subsystem for secure communication among participants without worrying about network intrusion threats. Alternatively, an even more trustworthy higher layer can be built on top of the communication subsystem—by incrementally using intrusion-tolerance mechanisms—such as a replication management protocol that's resilient against both network and host intrusions.

Architectural options

A MAFTIA host's structure relies on a few main architectural options, some of which are natural consequences of the discussions in the previous section:

- The notion of trusted—versus untrusted—hardware. Most of MAFTIA's hardware is untrusted, but small parts of it are trusted to the extent of some quantifiable measure of trustworthiness—for example, being tamper-proof by construction.
- The notion of trusted support software that can execute a few functions correctly, albeit in an environment subjected to malicious faults.

- The notion of a runtime environment that extends operating system capabilities and hides heterogeneity among host operating systems by offering a homogeneous API and framework.
- The notion of trusted distributed components, materialized by MAFTIA middleware, which are modular and multilayered. Each layer overcomes lower layers' faulty behavior.

We can depict the MAFTIA architecture in at least three different dimensions (see Figure 2). The *hardware* dimension includes the host and networking devices that compose the physical distributed system. Within each node, the operating system and runtime platform (which can vary from host to host) provide *local support* services. Finally, MAFTIA provides *distributed software*: the layers of middleware running on top of the runtime support both the mechanisms that each host provides and MAFTIA's native services—authorization, intrusion detection, and trusted third parties. To operate securely across several hosts even in the presence of malicious faults, applications running on MAFTIA use the abstractions that the middleware and application services provide.

Hardware

We assume that the hardware in individual MAFTIA hosts is untrusted in general. In fact, most of a host's operations run on untrusted hardware—such as the usual PC or workstation machinery connected through the normal networking infrastructure to the Internet, which we call the *payload channel*. However, some hosts might have pieces of hardware that are trusted to the extent of seeming tamper-proof (that is, we assume intruders don't have direct access to the inside of these components). MAFTIA features two incarnations of such hardware, both of which are easy to incorporate in standard machines because they're commercial-off-the-shelf (COTS) products. One is a *smart card* (actually a Java card), connected to the machine's hardware and interfaced by the operating system. The other is an *appliance board*, which has a processor and an adapter to a (trusted) control network. The runtime support has specialized functions provided by the trusted support software and implemented in two components, the Java Card Module (JCM) and the Trusted Timely Computing Base (TTCB). For less demanding configurations, we also designed a software-implemented TTCB.⁴

Local runtime support

The MAFTIA architecture's runtime support dimension essentially consists of the operating system augmented with appropriate extensions. The middleware, service, and application software modules run on the Java virtual machine (JVM) runtime environment. The JCM assists the operation of a reference monitor that supports the MAFTIA authorization service.⁵ This reference monitor checks all accesses to local objects and autonomously manages all access rights for local objects. We trust the Java card to be tamper-proof for application services whose value is much less than the effort—in means or time—necessary to subvert it.

Distributed runtime support

The TTCB is a distributed trusted component responsible for providing a basic set of trusted time and security services to middleware protocols for communication and activity support. The TTCB services are accessed locally through runtime support but can have global reach, such as making a value known to all local TTCB parts, thus limiting the potential for Byzantine faults by malicious protocol participants, as we discuss later. We can assume that the TTCB component is infeasible to subvert, but it might be possible to interfere with its software component interactions through the JVM. Although this exposes a local host to compromise, it doesn't undermine the distributed TTCB operation.

Middleware

The middleware layers implement functionality at different levels of abstraction and make it accessible at the interfaces of several middleware modules. These interactions occur through the runtime environment via predefined APIs.

As mentioned earlier, a middleware layer can overcome the fault severity at lower layers and provide certain functions in a trustworthy way. A (distributed) transactional service, for example, trusts that a (distributed) atomic multicast component ensures typical properties (agreement and total order), regardless of the fact that the underlying environment can suffer malicious Byzantine attacks.

MAFTIA's intrusion-tolerance strategies

Given the variety of possible MAFTIA applications, different architectural strategies should be available to cope with different risk scenarios. MAFTIA offers several intrusion-tolerance strategies through a versatile combination of admissible failure assumptions. System designers can apply these

strategies at several levels of abstraction in the architecture and, most important, in the implementation of the middleware and application services. An extended discussion appears elsewhere.⁶

Ultimately, MAFTIA supplies different solutions for different levels of threats and criticality (depending on the value of services or information), keeping the best possible performance-resilience trade-off. However, anything less than "arbitrary behavior" as an assumption raises eyebrows among many security and cryptography experts, so this statement deserves discussion.

A crucial aspect of any fault- or intrusion-tolerant architecture is the fault model on which the system architecture is conceived and component interactions are defined. Classically, making assumptions about hacker behavior isn't very sensible, which is why many system designers tend to assume any behavior is possible (asynchronous, arbitrary, or Byzantine).

However, such weak assumptions limit the system's power and performance—for example, could it still fulfill a service-level agreement (SLA), which is a contract a service provider makes with a client about quality of service (QoS)?

Architectural hybridization

Up until recently, increased system performance or QoS have meant less security. But MAFTIA has advanced the state of the art, demonstrating that it's possible to build applications that gather the best of both worlds: high resilience at the level of arbitrary failure systems and high performance at the level of controlled failure systems.

Through the innovative concept of *architectural hybridization*, the architecture simultaneously supports components with different kinds and severity of vulnerabilities, attacks, and intrusions.⁷ For example, part of the system might be assumed to be

subject to malicious attacks, whereas other parts are specifically designed in a different manner, to resist different sets of attacks. These hybrid failure assumptions are in fact enforced in specific parts of the architecture, by system component construction, and are thus substantiated. That is, in MAFTIA, trusting an architectural component doesn't mean making possibly naive assumptions about what a hacker can or can't do to that component. Instead, the component is specifically constructed to resist a well-defined set of attacks.

Wormholes model

Architecture isn't enough to solve the resilience problem, though. The correctness arguments of the algorithms and protocols rely on the *wormholes model*, a hybrid distributed-systems model that postulates the existence of enhanced parts (or wormholes) of a distributed system capable of providing stronger behavior than is assumed for the rest of the system. MAFTIA perfected this hybrid distributed-system model specifically for Byzantine faults.⁷

Protocol participants exchange messages in a world full of threats. If some of them are malicious and cheat, a wormhole can implement a degree of trust for low-level operations: as a local oracle whose information can be trusted, as a channel that participants can use to get in touch with each other securely (even for rare moments and for scarce bits of information), or as a processor that reliably executes a few specific functions or synchronization actions. Systems using strands of this model have received increasing amounts of attention lately.

A wormhole in the particular use of a trusted security component might look like a trusted computing base with a reference monitor. In fact, the concept is more general in two senses. First, trusted computing base's philosophy was based on system-level prevention of intrusions, whereas wormhole mod-

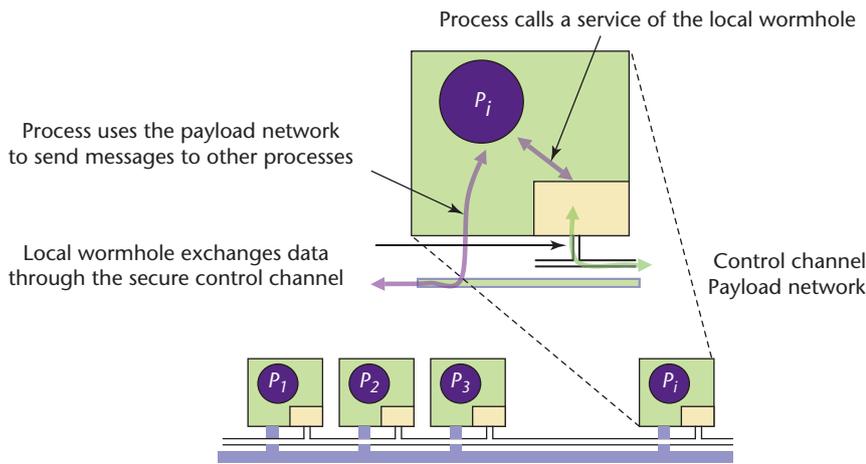


Figure 3. Architecture with a Trusted Timely Computing Base (TTCB) wormhole. The general systems are depicted in dark shading, and the wormhole is in light shading.

els have intrusion tolerance in mind: they would allow (and tolerate) intrusions even in the reference-monitor-protected part, significantly reducing the part of the system about which strong claims of tamper-proofness are made. Second, a wormhole can implement any semantics, including a reference monitor, but also simpler mechanisms, such as random number generators or key distribution, or innovative distributed agreement microprotocols.⁸

Real wormholes

The wormholes model can be realistically implemented via the notion of architectural hybridization.⁷ MAFTIA implements each wormhole as a trusted-trustworthy component—a component that can be trusted because it’s “better” by construction.

Figure 3 shows a snapshot of a real system that uses TTCB wormholes. The general, or payload, subsystems are the normal machines and networks depicted in dark shading in the figure. Each host contains the typical software layers such as the operating system, runtime environment, and middleware. Think of a small appliance board connected to the machine bus (these boards exist

as COTS components), implementing a set of useful functions in a protected manner. This is the MAFTIA TTCB wormhole, depicted with light shading in Figure 3. This proof-of-concept prototype was built using simple prevention techniques. However, adequately designed MAFTIA wormholes can withstand extreme attack levels, even life-cycle attacks (insertion of malicious code during development), by resorting to recursive design: a wormhole can itself be a modular or distributed subsystem designed with intrusion-tolerance techniques (replication, diversity, obfuscation, rejuvenation), to substantiate trustworthiness claims as firmly as desired, such as eliminating single points of failure.

As we mentioned earlier, wormholes can be local or distributed. The TTCB is distributed through a control channel, which is a private network. As a practical example, consider a Web server farm in a data center that’s tolerant to intrusions from the Internet: servers are connected through the payload Ethernet to the Internet, but the local wormhole boards are isolated from the directly attackable servers. The boards are interconnected through a secondary Ethernet that’s completely isolated from the payload

Ethernet or Internet—this is the private control channel. Even if the machine is corrupted, the hacker can’t tamper with the local wormhole board or with the control channel.

Trusted-trustworthy components

MAFTIA assumes a fairly severe fault model, assuming that hosts and the communication environment are asynchronous and can all be intruded upon. However, hosts can have local trusted components implementing certain functions (such as random number generation, signature, and time) that can be invoked at certain steps of the MAFTIA software’s operation and whose result can be trusted as always correct, regardless of intrusions in the rest of the system. The construction of the MAFTIA authorization service followed this local trusted components strategy, which is implemented around Java cards fitted in some hosts.⁵

The distributed trusted components strategy amplifies the scope of trust. As such, certain global actions can be trusted (such as global time and block agreement), despite generally malicious communication and host environments. MAFTIA implements this strategy through the TTCB, which is in effect a security kernel distributed across several hosts. Several of the MAFTIA middleware protocols follow this strategy, and in fact these protocols support the MAFTIA intrusion-tolerant transactional service.⁶

Arbitrary failure assumptions

The hybrid failure approach, however resilient, relies on trusted component assumptions, or trustworthiness. Several operations will have a value or criticality such that the risk of failure due to possible violation of these assumptions, however small, can’t be incurred.

The only way to lower the risk even further is by resorting to arbi-

rary failure modes, in which nothing is assumed about the way components could fail. Consequently, this is another strategy pursued in MAFTIA—arbitrary-failure-resilient components—namely, communication protocols of the Byzantine class that don't make assumptions about the existence of trusted components. Some of the protocols the MAFTIA middleware uses to follow this strategy are of the probabilistic Byzantine class and offer several qualities of service (binary and multivalued Byzantine agreement and atomic broadcast). Some MAFTIA trusted-third-party services rely on them.⁹

MAFTIA middleware

Figure 4 shows the MAFTIA middleware's layers. The lowest layer is the *multipoint network* (MN), which is created on the physical infrastructure. Its main properties are the provision of multipoint addressing, basic secure channels, and management communications, all of which hide the underlying network's specificities.

The *communication support services* (CS) module implements basic cryptographic primitives, Byzantine agreement, group communication with several reliability and ordering guarantees, clock synchronization, and other core services. The CS module depends on the MN module to access the network. The *activity support services* (AS) module implements building blocks that assist participant activity, such as replication management, leader election, transactional management, authorization, key management, and so forth. It depends on the services the CS module provides.

The block to the left of the figure implements failure detection and membership management. Failure detection assesses remote hosts' connectivity and correctness and local processes' liveness. Membership management, which depends on failure information, helps

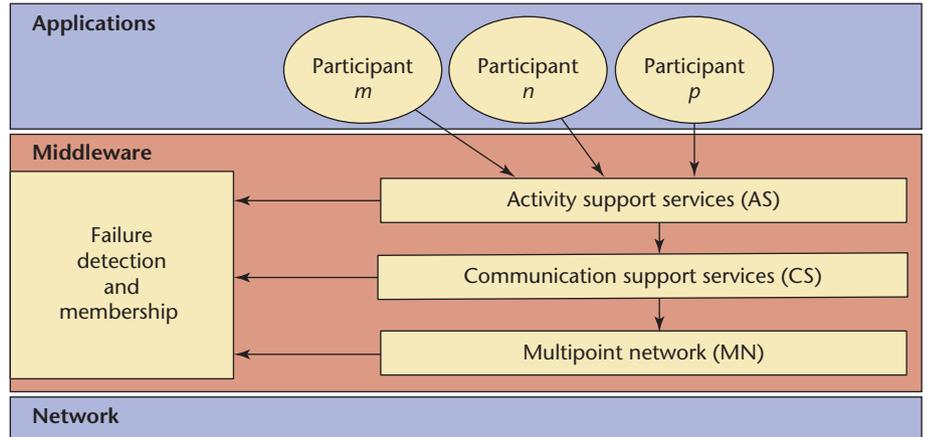


Figure 4. Detail of the MAFTIA middleware, showing the different architectural blocks.

create and modify group memberships (registered members) and the view (currently active, nonfailed, or trusted members). Both the AS and CS modules depend on this information.

As discussed earlier, an established way for achieving fault or intrusion tolerance is to distribute a service among a set of servers and then use replication algorithms for masking faulty servers. No single server has to be trusted completely, and the overall system derives its integrity from a majority of correct servers. Consequently, a very important part of the MAFTIA architecture is related to the algorithmic suites that implement communication and agreement among processes in different hosts.

Let's look more closely at the two main configurations of the MAFTIA middleware and algorithms.⁶

Byzantine agreement in an arbitrary world

In this configuration, the system model doesn't include timing assumptions and is characterized by a static set of servers with point-to-point communication and the use of modern threshold cryptography. There are no a priori trusted components, and trusted applications are implemented by deterministic

state machines replicated on all the servers and initialized to the same state. An atomic broadcast protocol delivers client requests, imposing a total order on all of them and guaranteeing that the servers perform the same sequence of operations. The atomic broadcast is built from a randomized consensus protocol—that is, a protocol for Byzantine agreement.

Model. This asynchronous model is subject to Fischer, Lynch, and Paterson's¹⁰ impossibility result of reaching consensus by deterministic protocols (FLP). Many developers of practical systems seem to have avoided this model in the past and built systems that are weaker than consensus and Byzantine agreement. However, randomization can solve Byzantine agreement in an expected constant number of rounds, as MAFTIA does. We use Byzantine agreement as a primitive for implementing atomic broadcast, which in turn guarantees a total ordering of all delivered messages.

Cryptography. To protect keys, we use threshold cryptography, an intrusion-tolerant form of secret sharing. Secret sharing lets a group of n parties share a secret such that t or fewer of them have no information about it,

but $t + 1$ or more can uniquely reconstruct it. However, someone can't simply share a cryptosystem's secret key and reconstruct it to decrypt a message because as soon as a single corrupted party knows the key, the cryptosystem becomes completely insecure and unusable.

In a *threshold public-key cryptosystem*, for example, each party holds a *key share* for decryption, which is done in a distributed way. Given a ciphertext resulting from encrypting a message, individual parties decrypting it output a decryption share. At least $t + 1$ valid decryption shares are required to recover the message. Another important cryptographic algorithm is the *threshold coin-tossing scheme*, which provides a source of unpredictable random bits that only a distributed protocol can query. It's the key to circumventing the FLP impossibility result, and the randomized Byzantine agreement protocol uses it in MAFTIA.

No timing assumptions. Working in a completely asynchronous model is attractive because the alternative of specifying timeout values actually constitutes a system's vulnerability. It's sometimes easier, for example, for a malicious attacker to simply block communication with a server than subvert it, but a system with timeouts would nevertheless classify the server as faulty.

This is how attackers can fool time- or timeout-based failure detectors into making an unlimited number of wrong failure suspicions about honest parties. The problem arises because a crucial assumption underlying the failure detector approach—namely, that the communication system is stable (failure detection is accurate) for a sufficiently long period to allow protocol termination—doesn't hold against a malicious adversary.

Secure asynchronous agreement and broadcast. Several protocols are used in this architecture configu-

ration, such as reliable and consistent broadcast, atomic broadcast, and secure causal atomic broadcast. Detailed descriptions appear elsewhere.^{9,11,12} These protocols work under the optimal assumption that fewer than one-third of the processes become faulty at any time. They're implemented in a modular and layered way.

Byzantine agreement requires all parties to agree on a binary value proposed by an honest party. Our randomized protocol checks if the proposal value is unanimous or else adopts a random value.¹¹ Multivalued Byzantine agreement is based on the previously described protocol and provides agreement on values from large domains.¹² A basic broadcast protocol in a distributed system with failures is reliable broadcast, which provides a way for a party to send a message to all other parties. It requires that all honest parties deliver the same set of messages and that this set includes all messages broadcast by such parties, but makes no assumptions if a message's sender is corrupted. An atomic broadcast guarantees a total order on messages such that honest parties deliver messages in the same order. Any implementation of atomic broadcast must implicitly reach agreement whether to deliver a message sent by a corrupted party, and, intuitively, this is where the Byzantine agreement module is needed: the parties proceed in global rounds and agree on a set of messages to deliver via multivalued agreement.¹² A secure causal atomic broadcast ensures a causal order among all broadcast messages. It's implemented by combining atomic broadcast with a robust threshold cryptosystem. Encryption ensures that messages remain secret up to the moment at which they're guaranteed to be delivered, preventing any violations of causal order by a corrupted party.

Reliable communication

Other MAFTIA middleware con-

figurations follow a strategy based on distributed trusted components and architectural hybridization. In this configuration we implemented several distributed protocols, such as reliable multicast, atomic multicast, and consensus. These protocols' correctness arguments rely on the wormholes model.

Model. In this configuration, a group of processes executes a protocol, as Figure 3 suggests. Processes run outside the wormhole (in the dark part) and communicate by sending messages through the payload network. At certain points of their execution, however, they can request trusted services from the wormhole by calling its interface.

The global system assumptions in this configuration are weak: the system is assumed to be asynchronous, and processes and communication can suffer Byzantine faults. Consequently, this model is also bound to the FLP impossibility result¹⁰ mentioned earlier. The wormholes model lets processes invoke functions that have enough power to circumvent the FLP impossibility, while still maintaining a generically weak and thus resilient model.^{7,13} In this case, the TTCB and the control channel provide timely (synchronous) execution and communication among TTCB modules. In practical implementations, these synchrony guarantees can be ensured (despite the rest of the system being completely asynchronous) because the wormhole has complete control over its resources. Furthermore, it's assumed to fail only by crashing: it either provides its services as expected, or it simply stops running.

Example TTCB wormhole services. In MAFTIA, the wormholes metaphor is materialized by the TTCB, whose most important services are the local authentication service, which makes the necessary initializations and authenticates the local wormhole component before

the process; the trusted time-stamping service returns timestamps with the current global time; and the trusted block agreement service applies an agreement function to a set of values and returns information about who proposed what.

Designing wormhole-aware protocols. In the project, we designed several protocols to form a coherent Byzantine-resilient communication suite, comprising reliable multicast,¹⁴ atomic multicast, simple,⁸ and vector consensus,¹³ and state machine replication management.¹⁵

A correct use of the wormholes principle mandates that most of the protocol execution occurs in the payload subsystem. The wormhole services are only invoked when there is an obvious trade-off between what is obtained from the wormhole service and the complexity or cost of implementing it in the payload subsystem.

As a quick example, let's assume a reliable broadcast execution. The protocol starts with the sender multicasting a message through the payload channel; now the message's integrity and reception must be checked. In classical protocols, this entails some complexity or delay because the sender might be malicious or the network might be attacked or have omission failures. Wormholes can help here: the sender and all recipients send a hash of the message to the wormhole, which runs a simple agreement on the hashes in its protected environment, returning to everyone the sender's hash as a result. If all goes well, the protocol terminates in an extremely quick manner. In case of faults, additional information returned by the wormhole allows fast termination after a few additional interactions.

One achievement related to our model is that most of the protocols have lowered known bounds on the required total number n of processes to tolerate a given number of Byzantine faults f , from $n > 3f$ to $n > f + 1$

for reliable multicast¹⁴ and $n > 2f$ for state-machine replication.¹⁵

Another achievement concerns performance and complexity. Despite maintaining Byzantine resilience and working on essentially weak arbitrary and asynchronous settings, the protocols—thanks to wormholes—exhibit unusually high performance and low complexity when compared to alternative implementations.

You can find a detailed description of MAFTIA's history at <http://istresults.cordis.lu/index.cfm/section/news/tpl/article/BrowsingType/Features/ID/69871>. □

Acknowledgments

MAFTIA was a project of the EU's IST (Information Society and Technology) program. The EC supported this work through project IST-1999-11583.

References

1. P. Verissimo, N.F. Neves, and M. Correia, "Intrusion-Tolerant Architectures: Concepts and Design," *Architecting Dependable Systems*, LNCS 2677, R. Lemos, C. Gacek, and A. Romanovsky, eds., Springer-Verlag, 2003, pp. 3–36.
2. D. Powell and R.J. Stroud, eds., *Conceptual Model and Architecture of MAFTIA*, Project MAFTIA Deliverable D21, Jan. 2003; www.maftia.org/maftia/deliverables/D21.pdf.
3. R. Stroud et al., "A Qualitative Analysis of the Intrusion-Tolerant Capabilities of the MAFTIA Architecture," *Proc. Int'l Conf. Dependable Systems and Networks (DSN 04)*, 2004, pp. 453–461.
4. M. Correia, P. Verissimo, and N.F. Neves, "The Design of a COTS Real-Time Distributed Security Kernel," *Proc. 4th European Dependable Computing Conf.*, Springer Verlag, 2002, pp. 234–252.
5. Y. Deswarte et al., "An Intrusion-Tolerant Authorization Scheme for Internet Applications," *Proc. Int'l Conf. Dependable Systems and Networks*, IEEE CS Press, 2002, pp. C1.1–C1.6.
6. P. Verissimo et al., "Intrusion-Tolerant Middleware: The MAFTIA Approach," tech. report DI/FCUL TR 04-14, Dept. of Informatics, Univ. of Lisbon, Nov. 2004.
7. P. Verissimo, "Uncertainty and Predictability: Can They Be Reconciled?" *Future Directions in Distributed Computing*, LNCS 2584, Springer-Verlag, 2003, pp. 108–113.
8. N.F. Neves, M. Correia, and P. Verissimo, "Solving Vector Consensus with a Wormhole," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 12, 2005, pp. 1120–1131.
9. C. Cachin and J.A. Poritz, "Secure Intrusion-Tolerant Replication on the Internet," *Proc. Int'l Conf. Dependable Systems and Networks*, IEEE CS Press, 2002, pp. 167–176.
10. M.J. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *J. ACM*, vol. 32, no. 2, 1985, pp. 374–382.
11. C. Cachin, K. Kursawe, and V. Shoup, "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography," *Proc. 19th ACM Symp. Principles of Distributed Computing*, ACM Press, 2000, pp. 123–132.
12. C. Cachin et al., "Secure and Efficient Asynchronous Broadcast Protocols," *Advances in Cryptology*, LNCS 2139, J. Kilian, ed., Springer-Verlag, 2001, pp. 524–541.
13. M. Correia et al., "Low Complexity Byzantine-Resilient Consensus," *Distributed Computing*, vol. 17, no. 3, 2005, pp. 237–249.
14. M. Correia et al., "Efficient Byzantine-Resilient Reliable Multicast on a Hybrid Failure Model," *Proc. 21st IEEE Symp. Reliable Distributed Systems*, IEEE CS Press, 2002, pp. 2–11.
15. M. Correia, N.F. Neves, and P. Verissimo, "How to Tolerate Half Less One Byzantine Nodes in Practical Distributed Systems," *Proc. 23rd IEEE Symp. Reliable Dis-*

tributed Systems, IEEE CS Press, 2004, pp. 174-183.

Paulo E. Veríssimo is a professor at the University of Lisbon Faculty of Sciences and director of the LASIGE research laboratory. His research interests include architecture, middleware, and protocols for distributed, pervasive, and embedded systems, in the facets of real-time adaptability, security, and fault and intrusion tolerance (<http://navigators.di.fc.ul.pt/>). Contact him via www.di.fc.ul.pt/~pju.

Nuno F. Neves is an assistant professor at the University of Lisbon, Portugal. His research interests include security and fault tolerance in parallel and distributed systems. Neves has a PhD in computer science from the University of Illinois at Urbana-Champaign. He is a member of the IEEE. Contact him via www.di.fc.ul.pt/~nuno.

Christian Cachin is a research staff member at the IBM Zurich Research Lab-

oratory. His research interests include cryptographic protocols, security in distributed systems, and steganography. Cachin has a PhD in computer science from ETH Zürich. He is a member of the IACR, ACM, and IEEE. Contact him via www.zurich.ibm.com/~cca.

Jonathan Poritz is a lecturer at Colorado State University. His research interests include mathematical cryptography, quantum computing, privacy technology, trusted computing, and computer visualization of geometry. Poritz has a PhD in mathematics from the University of Chicago. He is a member of the ACM and the American Mathematical Society. Contact him via www.poritz.net/jonathan.

David Powell is a "directeur de Recherche CNRS" at LAAS-CNRS. His research interests include distributed algorithms for software-implemented fault-tolerance, ad hoc networked systems, and critical autonomous robotic systems. Powell has a PhD in computer science from the Toulouse National Poly-

technic Institute. Contact him at David. Powell@laas.fr.

Yves Deswarte is a "directeur de recherche" at the CNRS Laboratory for Analysis and Architecture of Systems. His research interests include fault-tolerance, security, and privacy in distributed computing systems. Contact him at Yves. Deswarte@laas.fr.

Robert Stroud is a reader in computing science at the University of Newcastle upon Tyne. His research interests include security and fault-tolerance. Stroud has a PhD in computing science from University of Newcastle upon Tyne. Contact him via www.cs.ncl.ac.uk/people/r.j.stroud/.

Ian Welch is a lecturer at Victoria University, New Zealand. His research interests include secure auctions, intrusion detection, aspect-oriented development, and community computing. Welch has a PhD from the University of Newcastle upon Tyne. Contact him at ian@mcs.vuw.ac.nz.

ADVERTISER / PRODUCT INDEX JULY/AUGUST 2006

Advertiser	Page Number	Advertising Personnel	
Addison Wesley	5	Marion Delaney IEEE Media, Advertising Director Phone: +1 415 863 4717 Email: md.ieeemedia@ieee.org	Sandy Brown IEEE Computer Society, Business Development Manager Phone: +1 714 821 8380 Fax: +1 714 821 4010 Email: sb.ieeemedia@ieee.org
ISSE 2006	11	Marian Anderson Advertising Coordinator Phone: +1 714 821 8380 Fax: +1 714 821 4010 Email: manderson@computer.org	
LinuxWorld Conference & Expo 2006	13		
Unix Security Symposium 2006	Cover 4		
<i>Boldface denotes advertisements in this issue.</i>			
Advertising Sales Representatives			
Mid Atlantic (product/recruitment) Dawn Becker Phone: +1 732 772 0160 Fax: +1 732 772 0161 Email: db.ieeemedia@ieee.org	Midwest (product) Dave Jones Phone: +1 708 442 5633 Fax: +1 708 442 7620 Email: dj.ieeemedia@ieee.org Will Hamilton Phone: +1 269 381 2156 Fax: +1 269 381 2556 Email: wh.ieeemedia@ieee.org Joe DiNardo Phone: +1 440 248 2456 Fax: +1 440 248 2594 Email: jd.ieeemedia@ieee.org	Midwest/Southwest (recruitment) Darcy Giovingo Phone: +1 847 498-4520 Fax: +1 847 498-5911 Email: dg.ieeemedia@ieee.org	Northwest/Southern CA (recruitment) Tim Matteson Phone: +1 310 836 4064 Fax: +1 310 836 4067 Email: tm.ieeemedia@ieee.org
New England (product) Jody Estabrook Phone: +1 978 244 0192 Fax: +1 978 244 0103 Email: je.ieeemedia@ieee.org	Southeast (recruitment) Thomas M. Flynn Phone: +1 770 645 2944 Fax: +1 770 993 4423 Email: flynttom@mindspring.com	Southwest (product) Steve Loerch Phone: +1 847 498-4520 Fax: +1 847 498-5911 Email: steve@didierandbroderick.com	Japan Tim Matteson Phone: +1 310 836 4064 Fax: +1 310 836 4067 Email: tm.ieeemedia@ieee.org
New England (recruitment) John Restchack Phone: +1 212 419 7578 Fax: +1 212 419 7589 Email: j.restchack@ieee.org	Southeast (product) Bill Holland Phone: +1 770 435 6549 Fax: +1 770 435 0243 Email: hollandwfh@yahoo.com	Northwest (product) Peter D. Scott Phone: +1 415 421-7950 Fax: +1 415 398-4156 Email: peterd@pscottassoc.com	Europe (product) Hilary Turnbull Phone: +44 1875 825700 Fax: +44 1875 825701 Email: impress@impressmedia.com
Connecticut (product) Stan Greenfield Phone: +1 203 938 2418 Fax: +1 203 938 3211 Email: greenco@optonline.net		Southern CA (product) Marshall Rubin Phone: +1 818 888 2407 Fax: +1 818 888 4907 Email: mr.ieeemedia@ieee.org	

Part Algo – APPENDIX
(Resilience Algorithms and Mechanisms)

Fighting Erosion in Dynamic Large-Scale Overlay Networks

Abstract

Overlay management protocols have been introduced to guarantee overlay network connectivity in dynamic large-scale peer-to-peer systems. Some of these protocols have been specifically designed to avoid the partitioning of the overlay in large clusters (network breakage) despite massive node failures and the continuous arrivals/departures of nodes (churn). In this paper we identify a second effect connected to churn, namely network erosion. We show how erosion affects overlay network connectivity and point out that even a strongly connected overlay network, when exposed to continuous churn, can be disintegrated in a relatively short time. More specifically the consequences of erosion are shown, through an experimental study, in the context of overlay management protocols based on view-exchange. We finally propose a connection recovery mechanism to be endowed at each node which is able to collaboratively detect node isolation and the presence of small clusters. This mechanism is shown to be effective in reducing the erosion of an overlay network exposed to continuous churn and to quickly recover its connectivity as soon as churn ends.

1. Introduction

In the last decade the advent of peer-to-peer (p2p) computing introduced a new model of distributed computation where (i) the scale of the system can be very large, comprising up to million of users (peers), (ii) each peer acts independently from all the others, actually precluding any form of centralized network-wide administration or management, (iii) each peer acts as a client of the service and cooperates with other peers to enable services for other participants, and (iv) the system, due to its size and the autonomy of each peer, is intrinsically dynamic as peers can join in or leave at any time.

In this context the basic problem that must be solved in order to build distributed applications is how to guarantee connectivity among participants. Connectivity is, in fact, the basic building block to enable network communications among peers. Modern p2p systems use, to this aim, an

overlay network, i.e. a logical network connecting all the participants, whose maintenance is demanded to a specific protocol, namely an *Overlay Management Protocol* (OMP).

Motivation. When p2p systems grow up to very large scales, phenomena connected to the dynamic behaviour of nodes grow in importance: the continuous arrival and departure of nodes, usually known as *churn*, can cause, if not properly addressed, *overlay network partitioning*.

To this regard, OMPs based on gossip approaches [5, 6, 3, 8] revealed to be very effective in the prevention of *large overlay network breakages*, i.e. the partitioning of the overlay network in two (or more) clusters of approximately the same size. Overlay network breakages can be considered as catastrophic events that affect the system with a large and abrupt reduction of the overlay network connectivity. These protocols aim to build and maintain, through some lightweight mechanisms, an overlay network with a random topology; the random topology is used to guarantee a low probability of overlay network breakage even when a very large portion of nodes is abruptly removed.

However overlay network partitioning can also take the form of a second distinct effect (beside network breakage): *network erosion*. Network erosion is a subtler, endemic phenomenon, caused even by low churn rates, which progressively remove single nodes or tiny clusters from the frontier of the main overlay network cluster. This frontier is constituted by those nodes whose neighbors have been removed, and whose views contain dangling edges. These nodes are indeed weakly connected to the network and further node removals can bring them to a state of complete isolation. Contrarily to network breakages, erosion progressively affects overlay network connectivity.

Contribution. Many OMPs underestimate this problem simply assuming that an isolated node could eventually rejoin the overlay [3] but without employing specific mechanisms. In this paper we *point out that fighting network erosion deserves the same attention as network breakages*. Erosion can be indeed so disruptive that a strongly connected overlay network exposed to continuous churn can be quickly led to its complete disintegration. More specifically we show through an experimental study how badly

network erosion affects view-exchange based overlay management protocols, like Cyclon [8] and ADH [3]. Overlay networks built through these protocols are, in fact, progressively eroded till their complete disgregation, regardless of the actual churn rate, as long as the churn period lasts enough (i.e. severe churn rates for short periods of time show the same erosion effect as lower rates for longer periods). Our study thus confirms that these OMPs were not designed to take erosion into account, and are thus not able to face its effects.

To fight erosion we propose a *connection recovery* mechanism, whose goal is twofold: (i) increase robustness of the overlay network during long periods characterized by churn and (ii) recover connectivity during periods when churn is absent. To reach these goals we exploit an intelligent re-join method that locally detects a status isolation from the network's main cluster. The presence of small clusters is also recognized and addressed by leveraging collaboration among nodes. Our experimental studies show how the connectivity recovery mechanism is able to reduce the effects of network erosion during long periods of time characterized by churn, increasing the capacity of the OMPs to quickly react to topology changes; the experiments also show that, through this mechanism, the overlay network is able to quickly regain full connectivity when the system undergoes a stability period without churn.

Related work. The effects of network erosion have never emerged clearly in other works on OMPs [3, 8, 5, 6]. This is due to the fact that each OMP has focussed on different goals, like obtaining a good tradeoff between network size and view size or building overlay networks with low diameters. Nevertheless, none of them has been exposed to continuous churn. For example Voulgaris et al. in [8] and Ganesh et al. in [6] analyzed protocols' behaviour only in a static setting without churn. Allavena et al. in [3] and Eugster et al. in [5] addressed the overlay network partition problem considering prevention and recovery from large network breakages. Specifically the former provides a completely decentralized solution while the latter assumes the presence of a set of fixed nodes in the system. Nevertheless both solutions have not been experimentally evaluated, thus the effects of overlay network erosion did not come out. Finally the first analysis of the behaviour of two different OMPs, namely SCAMP [6] and Cyclon [8], under continuous churn has been presented in [4]. In that work some problems related to continuous churn have been pointed out, but the paper did not provide any solution to address them.

The remainder of the paper is organized as follows. Section 2 introduces the reader to the details of two OMPs: Cyclon and ADH. Section 3 shows how these two OMPs are

brought to their knees from continuous churn. Section 4 introduces the *connection recovery mechanism*, while Section 5 shows its effectiveness through an experimental study. Finally Section 6 concludes the paper.

2. View Exchange-Based Overlay Maintenance Protocols

An overlay network is a logical network built on top of a physical one (usually the Internet), by connecting a set of nodes through some links. A distributed algorithm running on nodes, known as the Overlay Maintenance Protocol (OMP), takes care of the overlay "healthiness" managing these logical links. The common characteristic of all OPMs is that each node maintains links to other nodes in the system. This set of links is limited in its size in order to favour system scalability and it is usually known as the *view* of the node. Views construction and maintenance should be such that the graph, obtained by interpreting links in views as arcs and nodes as vertexes, is connected, as this is a necessary condition to enable communication from each node to all the others.

OMP differentiate among themselves with respect to the technique they employ to build and maintain views. They can be divided in two broad groups basing on the strategy used to manage node leaves:

Reactive Protocols - are run only when a node decides to leave the system [6].

Proactive Protocols - continuously adjust the network topology in order to allow nodes to leave without executing any specific algorithm [8, 3].

In this paper we refer to the latter group as OMPs pertaining to it are considered more suited to dynamic environments like the one we refer to. In the following we will analyze the two cited OMPs. Both are based on a technique known as *view exchange* that requires node to continuously exchange part of their views in order to keep the overlay topology as close as possible to a *random graph*. Random graphs are characterized by strong connectivity, a property that is exploited to avoid network partitioning: node leaves and faults can, in fact, be simply ignored by the OMP as the random topology is supposed to remain connected despite node removals.

2.1. Cyclon

Cyclon [8] follows a proactive approach where nodes perform a continuous periodic view exchange activity with their neighbors in the overlay. The view exchange phase (named in this case "shuffle cycle") aims at randomly mixing views between neighbor nodes. Joins are managed in a

reactive manner, through a join procedure, while voluntary departures of nodes are handled like failures (no leave algorithm is provided). A simple failure detection mechanism is provided in order to clean views from failed nodes.

Data Structures - Each node maintains only a single view of nodes it can exchange data with. The size of the view is fixed and can be set arbitrarily. Each node in the view is associated to a local age, indicating the number of shuffle cycles during which the node was present in the view.

Join Algorithm - A node A joins by choosing one node (*bootstrap node*) among those already present in the network. The protocol starts then a set of independent random walks from the bootstrap node. The number of random walks is equal to the view size, while the number of steps per each random walk is a parameter of the algorithm. When each random walk terminates, the last visited node, say B , adds A to its view by replacing one node, say C , which is added to A 's view using an empty slot.

Shuffle Algorithm - The shuffle algorithm is executed periodically at each node. A shuffle cycle is composed of three phases. In the first phase a node A , after increasing the age of all the nodes in its view, chooses its shuffle target, B , as the node with higher age among those in its view. Then, A sends to B a shuffle message containing $l - 1$ nodes randomly chosen in A 's view, plus A itself. In the second phase, B , once received the shuffle message from A , replaces $l - 1$ nodes in its view (chosen at random) with the l nodes received from A and send them back to A . In the final phase A replaces the nodes previously sent to B with those received from it. Overall, the result of one shuffle cycle is an exchange of l links between A and B . The link previously connecting A to B is also reversed after the shuffle.

Handling Concurrency - In the specifications given in [8], no action was defined in the scenario of two (or more) *concurrent* shuffle cycles, e.g. when a node A , during a shuffle cycle in progress with B , is selected as a target node by receiving a shuffle message from C . If concurrency is considered, the nodes sent by A to B can be modified by the concurrent shuffle involving A and C . To analyze the behaviour of Cyclon in concurrent scenarios we extended the original specification in order to address this situation: when nodes that should be replaced by A are no longer present in its cache, A replaces some nodes chosen at random.

2.2. ADH

The OMP proposed in [3], to which we will refer with the name ADH, employs a slightly different strategy to maintain views. Each node periodically substitutes its whole view with a new one, which is built basing on information collected since the last view change. Even in this case joins are managed in a reactive manner, through a join procedure, while voluntary departures of nodes are

handled like failures. Failure detection techniques are not used because crashed nodes are automatically discarded by the view change algorithm as time passes by.

Data Structures and Parameters - As Cyclon, also ADH employs a single view for each node. The size of the view k is fixed and can be set arbitrarily. Two more parameters are used: the *fanout* f and the *weight of reinforcement* w . Both are detailed later in this section.

Join Algorithm - Nodes joining the overlay network fill their initially empty views with the view of one of the nodes already in the system. ADH does not prescribe any specific method to choose this *bootstrap node*, as the OMP should be always able to balance the network (approximating a random topology). This is an important characteristic because a non-random choice of the bootstrap node is a problem that, if not addressed, can potentially lead to the construction of networks whose topologies are far from being random [7].

View Change Algorithm - Each node updates its view periodically, at the end of every *round*¹. During a round each node collects:

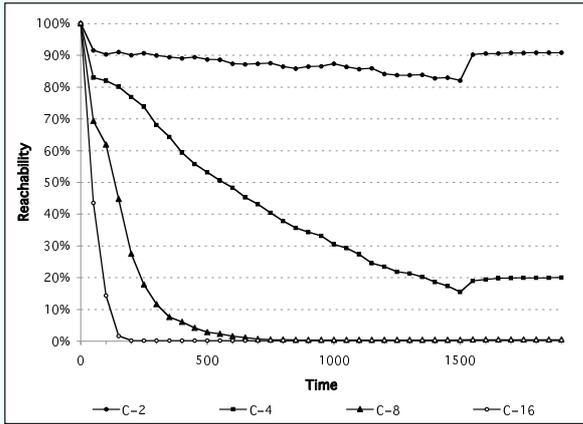
- a list L_1 comprising the local views of f nodes chosen at random from its view;
- a list L_2 comprising those nodes that requested its view during the round.

At the end of each round these two lists are used to create the local view that will be used in the next round. The new view is built by choosing k nodes from both L_1 and L_2 . The weight of reinforcement w ($w \in [0, \infty]$) is used to decide from which list a node must be picked: if $w = 0$ then all nodes are selected in L_1 , if $w = 1$ nodes are selected with equal probability in L_1 and L_2 , and, finally, if $w = \infty$ then all nodes are selected in L_2 . This mechanism is used to keep the network "clean" of crashed nodes (that surely will not appear in L_2), while mixing views. For these reasons the authors of [3], with respect to the value of w , suggest "Larger is better and will be either 1 or ∞ on a typical implementation".

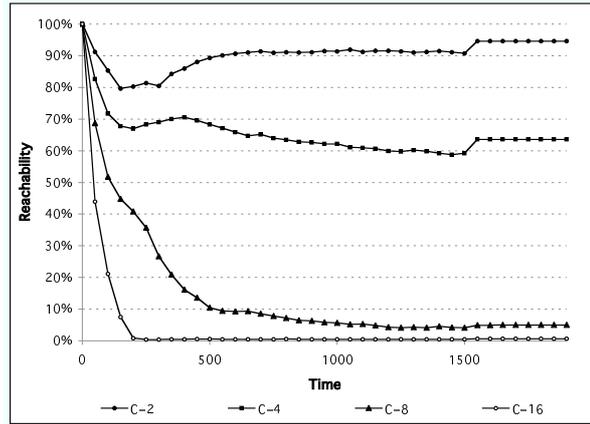
3. Overlay Robustness Under Continuous Churn

The protocols presented in Section 2 are able to build overlay networks whose topology approximates a random graph [8, 3]. Thanks to this property, systems built with these OMPs are supposed to be highly resilient to node removals. In a dynamic p2p scenario participants can enter or

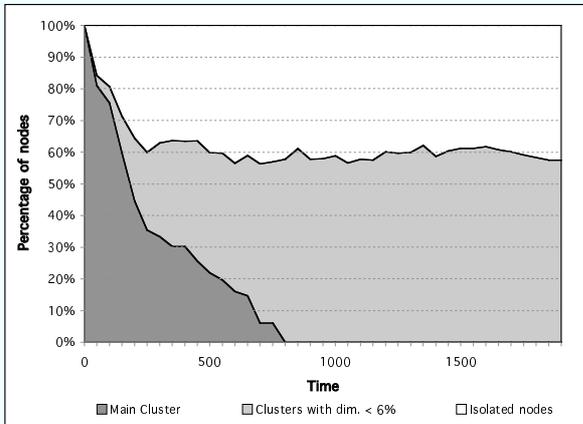
¹In [3] the protocol is introduced in a synchronous environment where the notion of *round* is clearly defined. In an asynchronous setting, like the one we used to test the algorithm, the notion of round can be approximated with the time lapse between two consequent view change operations. In our setting rounds pertaining to different nodes are not synchronized.



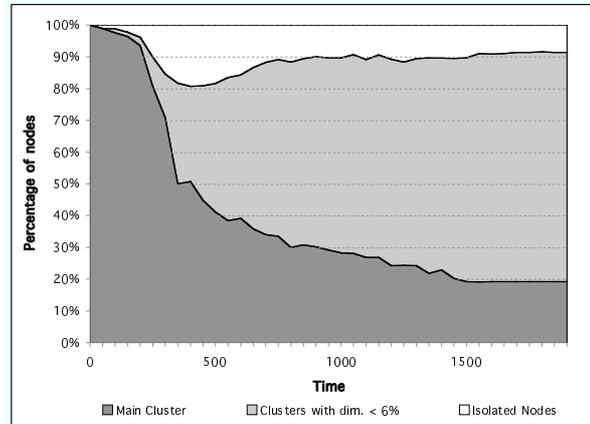
(a) Cyclon: evolution of R with different churn rates.



(b) ADH: evolution of R with different churn rates.



(c) Cyclon: evolution of overlay clustering with $C = 8$.



(d) ADH: evolution of overlay clustering with $C = 8$.

Figure 1. Evaluation of Cyclon and ADH under continuous churn.

leave the system at their will, at any time. The global rate at which these actions occur is called the *churn rate*. Node removals happen continuously during some time periods, i.e. churn is not an instantaneous phenomenon but its effects are rather durable in time. During churn periods characterized by sustained rates the time available to the OMP to repair the overlay network after a node departure, substituting dangling edges with valid ones, can become too short. Continuous churn can in these cases lead to *overlay network partitioning*, an event that manifests itself in two ways: *major network breakages* and *network erosion*.

Network breakages are caused by the removal of one or more nodes which form the common frontier of two otherwise independent large clusters. When these nodes are

removed no valid link exists that connects two clusters, thus nodes pertaining to distinct clusters cannot communicate. The net effect of a large network breakage is an abrupt and dramatic diminish of the overlay network connectivity. OMPs based on gossip approaches like Cyclon and ADH revealed to be very effective in the prevention of such events. The random graphs they build and maintain, in fact, guarantee that, even when a very large number of nodes are removed (independently from the speed at which these removal happens), the probability of a major network breakage is extremely low. This result is clearly stated in [8], where it is actually tested only in static scenarios where nodes are removed all at once, and in [3].

Network erosion is a subtler phenomenon, endemic in

systems affected by continuous churn, which progressively removes single nodes or tiny clusters from the frontier of the main network cluster. This frontier is constituted by those nodes whose neighbors have been progressively removed, and whose views contain dangling edges. These nodes are indeed weakly connected to the main cluster and further neighbor removals can quickly bring them to a state of complete isolation. It's important to note that erosion, contrarily to network breakages, is a phenomenon which affects progressively the overlay network, but whose effects are nevertheless dramatic. If not properly addressed erosion can, in fact, lead to the complete disgregation of the overlay network, quickly reducing a strongly connected cluster to a "dust" of isolated nodes.

To analyze the network erosion effect and how OMPs behave when it is present, we implemented and tested in a simulated environment (provided by Peersim [1]) both Cyclon and ADH. A run of each protocol was divided in three distinct periods: *creation*, *churn* and *stability*. During the creation period nodes join the system until a predefined network size N is reached. Neither leaves nor overlapped join operations occur during this phase. During the churn period, nodes continuously join and leave the network at a given *churn rate* C , i.e. at each unit of time, C new nodes invoke the join operation while C nodes in the overlay invoke the leave operation². The churn period ends after 1500 time units. At the end of this period the number of nodes in the overlay network is still N , while the total number of nodes that joined/left the system depends on the specific churn rate C . N was set to 1000 in all experiments³. Message transmission delay varies uniformly at random between 1 and 10 time units. 10 independent runs were made for each experiment.

Figures 1(a) and 1(b) shows overall reachability R , i.e. the average percentage of nodes that can be reached from any node in the overlay. This metric is strictly related to the connectivity of the overlay network, as any value lower than 100% indicates that at least one node cannot be reached by at least one other node. We evaluated the effect of the variation of the churn rate C ($C=2,4,8,16$) on R along the time (contiguous sampled points are separated by 100 join and 100 leave invocations). Both protocols have been evaluated with leaving nodes chosen uniformly at random in all the experiments. The curves show that reachability is strongly affected during the churn period. At the beginning the curves undergo a steep descending slope that is mainly due to the join of new nodes that are immediately considered as part of the system even if their views are initially empty; these nodes will affect negatively R until their

²This rather simple churn model was chosen to maintain the system at a constant size during tests.

³Further experiments, not reported here, show that the total number N of nodes in the system does not influence the final results as long as the ratio C/N is kept constant.

join procedures end filling their views. During the churn period network erosion continuously affect the overlay network isolating single nodes: this effect causes the continuous diminishment of R . It is important to note that this continuous diminishment of R is present even when the churn rate is relatively low (i.e. $C = 2$). At the end of the churn period the system regains a small percentage of reachability: the effect is mainly due to join procedures that end correctly after during the first part of the stability period. The interesting point is that, nevertheless, both OMPs are not able to regain full connectivity after the churn period ends: after a short period of time, used by the OMP to "heal" what remains of the original network, the system stabilizes to a constant reachability value that is always under 100%. This problem can be clearly imputed to isolated nodes whose messages are unable to reach any destination.

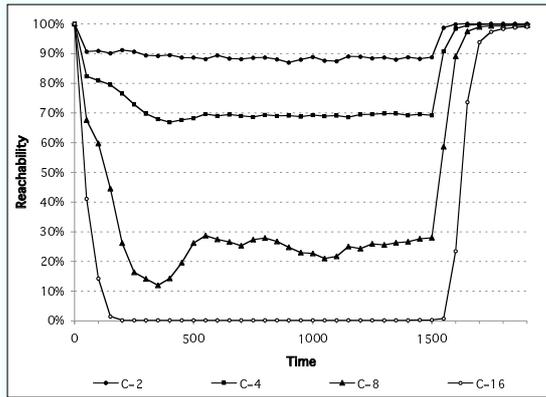
Figure 1(c) and 1(d) confirm this result showing the evolution of node clustering for experiments conducted with $C = 8$. The curves represent percentage of nodes pertaining to the largest cluster (dark grey area), to clusters smaller than 6% of the whole network (light grey area), and isolated nodes (white area). As the curves show a strong role is played by nodes that become isolated due to churn. What the curves do not show is that the great majority of small clusters are actually formed by new nodes that join the overlay starting from an isolated bootstrap node. From this point of view is fair to say that the analyzed OMPs are actually able to avoid large network breakages (we did not detect any massive network breakage during our tests), but node isolation, due to progressive network erosion, occurs very frequently.

4. Connection Recovery

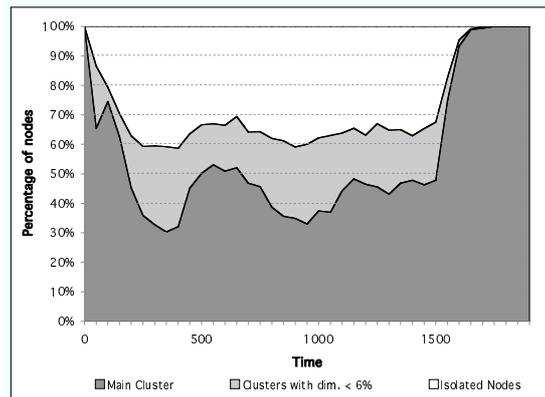
As we showed in the previous section, node isolation occurs quite frequently when the overlay network experiences erosion due to continuous churn. The authors of [8] and [3] did not address explicitly this problem, but without any intervention isolated nodes will remain endlessly in this state.

In this section we introduce a *connection recovery* mechanism that can be added to both OMPs (and, generally speaking, to every OMP for unstructured overlay networks). Aim of this mechanism is to let nodes detect their isolation state and act consequently in order to regain connection to the main cluster of the overlay network. Moreover our mechanism exploits cooperation among nodes to detect the presence of tiny clusters.

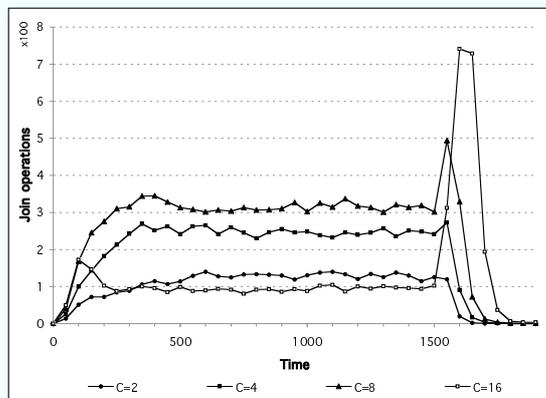
The basic idea of the connection recovery mechanism is simple: when a node detects that all the links in its partial view represent dangling edges, it triggers a new join procedure to regain connection to the system. In this way, isolated nodes will eventually re-join the system. Dead link



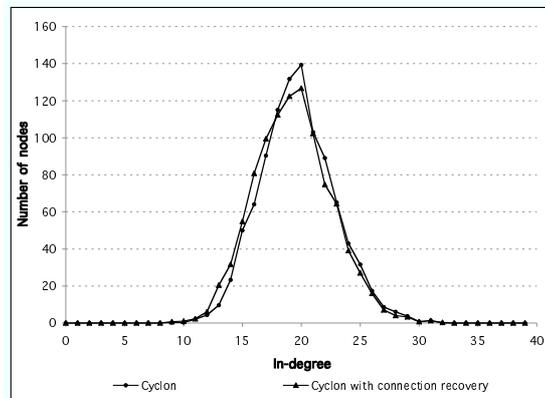
(a) Cyclon with connection recovery: evolution of R along time with different churn rates.



(b) Cyclon with connection recovery: evolution of overlay clustering with $C = 8$.



(c) Cyclon with connection recovery: evolution of re-joins.



(d) Cyclon with connection recovery: indegree distribution at the end of the simulation.

Figure 2. Evaluation of Cyclon with connection recovery.

detection is not done through some active mechanism but it is rather an indirect result of failed view exchanges (shuffles) with nodes that left the system. To treat also nodes pertaining to small clusters (that will not satisfy the condition expressed above), we added a cooperative aspect to the basic mechanism. When a node n tries to re-join the system it is assigned⁴ a *bootstrap* node n' , that is initially pinged to know the amount of links it has in its partial view. If this amount is under a predefined constant P , n rejects n' as a bootstrap node, asking for a new one. Nevertheless, n puts n' in a *low-connection list*: as soon as n encounters a node n^* which is able to guarantee a number of links larger

⁴This assignment can be realized exploiting the same method used for the plain Cyclon implementation.

than P , n warns all the nodes in its low-connection list that such a node exists in the overlay network. Note that n does not inform nodes in the list about the identity of n^* , otherwise a local star-like topology would be created around n^* . Nevertheless this sort of “signal” sent to some nodes will be interpreted by them as a clue that they are possibly part of either a small isolated cluster or a loosely connected part of the overlay: in consequence of this fact each node can independently decide to try a re-join even if its partial view is not empty. This decision is taken just looking at the current status of the view: if it contains a number of links still lower than P then the node will try to re-join the system. The parameter P actually influences the speed at which nodes re-join after detecting their isolation status. A possible solution to network partitioning based on connection recovery

is also suggested in [3]. In this case the authors propose, to detect the presence of small clusters, a completely local approach where each node just check the variance of nodes in its view: if this variance is very low then the probability of the node being stuck in an isolated small cluster is high. This proposal was not evaluated in [3], is thus hard to compare its effectiveness versus our connection recovery mechanism.

5. Evaluation

In this section we evaluate the connection recovery mechanism applied to the Cyclon OMP. The simulation setting is the one introduced in Section 3.

We first tested the evolution of reachability R and node clustering, the same test whose results related to the plain Cyclon OMP are reported in figures 1(a) and 1(c). Figure 2(a) shows that, thanks to our mechanism, reachability R remains consistently higher with respect to the values shown in figure 1(a). These curves point out a duplex effect caused by the connection recovery mechanism: the overlay network is maintained more connected during churn periods (at least for $C = 2, 4, 8$), as proven by the higher level of all the curves, and, as soon as the stability period starts, full connectivity is quickly regained, regardless of the R 's values previously reached. It is worth noting that reachability values for $C = 2, 4, 8$ remain almost constant (or raise) as soon as the connection recovery mechanism starts to work at its full potential. The time needed for this to happen is proportional to both P and the view size: the mechanism will, in fact, take up to *viewsize* shuffle intervals before starting the re-join for an isolated node. The same results are confirmed by Figure 2(b) where the evolution of overlay clustering is shown. The curves highlight how the connection recovery mechanism quickly pushes isolated nodes to rejoin the system, increasing the size of the largest cluster. Even the percentage of small clusters rapidly decreases thanks to the cooperative mechanism previously described.

Given the self-healing capability brought in by the connection recovery mechanism, an important aspect that must be evaluated is the capability of the mechanism to converge to a stable state when churn is absent. In order to show this, we measured the evolution of the number of join operations induced by connection recovery. As Figure 2(c) shows, the rate of join operations remains almost constant during the churn period, then immediately experiences a peak that is mostly due to the remaining isolated nodes that altogether try to re-join the system. As soon as the number of isolated nodes falls to zero the join rate drops. This actually means that the connection recovery mechanism remains active only for a limited time frame that is mainly linked to the length of the churn period, without causing further overhead when the system is stable.

Finally we wanted to test if and how the addition of the connection recovery mechanism can alter the behaviour of the original OMP in terms of the type of network topology built. Our mechanism actually only trigger automatically node leave and join procedures, thus we expected no differences on the "quality" of the network built. This idea is confirmed by the curves shown in figure 2(d) where we report the in-degree distribution of nodes at the end of the simulation, for Cyclon with and without our connection recovery mechanism. The tests for this latter version were conducted in a scenario with $C = 8$. The curves clearly show that both implementations are able to build overlay networks with the same in-degree distribution, proving that connection recovery does not alter in any way the fundamental characteristics of the overlay.

6. Conclusion

This paper pointed out the importance of continuous churn as the first class enemy that must be fought in order to maintain an overlay network connected. This has been done by analyzing two gossip-based overlay management protocols based on view exchange, namely Cyclon and ADH. While these protocols are effective in avoiding large network breakages, under continuous churn we showed that they suffer network erosion, i.e., single nodes or tiny clusters that are progressively detached from the main component of the overlay network.

Through an experimental study we showed how disruptive network erosion can be, up to the point where an overlay network maintained by a specialized protocol can be quickly digregated in a "dust" of isolated nodes or tiny clusters. Thus we proposed a connection recovery mechanism to be endowed at each node whose aim is to reduce the effect of network erosion under continuous churn and to completely recover overlay network connectivity as soon as churn ceases (even starting from a completely digregated network).

Even though these results are encouraging there are still various aspects that deserve further investigation. First of all the experiments we conducted are based on a simplified churn model: more realistic models, derived from real p2p systems logs, needs to be defined. Moreover, the presented connection recovery mechanism is based on collaboration among nodes. Other approaches can be investigated to take into account p2p environments where nodes behave selfishly.

References

- [1] Peersim. <http://peersim.sourceforge.net/>.
- [2] A. Allavena. *On The Correctness of Gossip-Based Membership Protocols*. PhD thesis, Cornell University, January 2006.

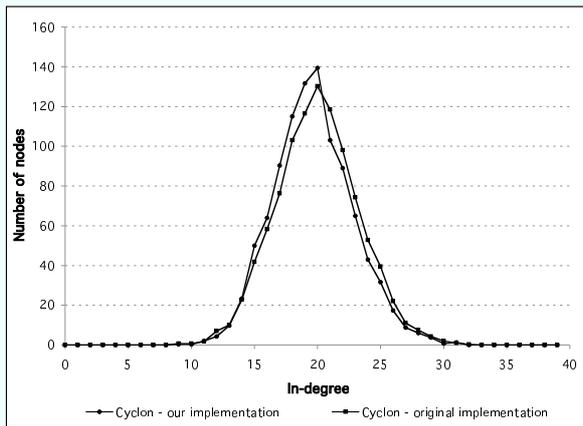


Figure 3. Cyclon: degree distribution of both our implementation and the original one

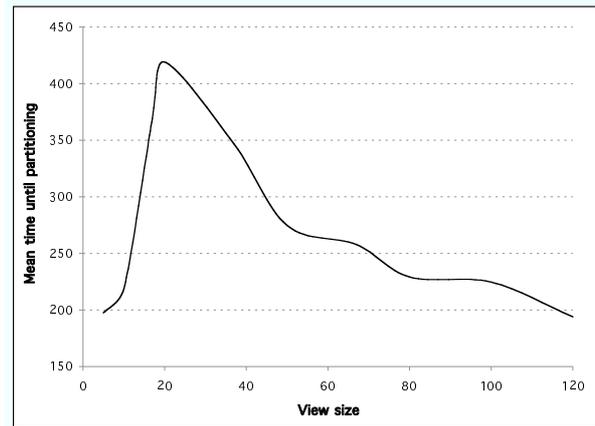


Figure 4. ADH: mean time until partitioning versus view size in our implementation.

- [3] A. Allavena, A. Demers, and J. E. Hopcroft. Correctness of a Gossip Based Membership Protocol. In *Proceedings of the 24th ACM annual symposium on Principles of Distributed Computing (PODC05)*, pages 292–301, 2005.
- [4] R. Baldoni, S. Bonomi, L. Querzoni, A. Ripa, S. T. Piergiovanni, and A. Virgillito. Evaluation of Unstructured Overlay Maintenance Protocols under Churn. In *Technical Report - MIDLAB 2/06*, Dip. Informatica e Sistemistica, Università di Roma "La Sapienza", 2006, (to appear) *International Workshop on Dynamic Distributed Systems (IWDDS)* (<http://www.dis.uniroma1.it/midlab/articoli/BBRQTV06IWDDS.pdf>), 2006.
- [5] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight Probabilistic Broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.
- [6] A. Ganesh, A. Kermarrec, and L. Massoulié. Peer-to-Peer Membership Management for Gossip-based Protocols. *IEEE Transactions on Computers*, 52(2):139–149, 2003.
- [7] M. Jelasity, R. Guerraoui, A. Kermarrec, and M. van Steen. The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-based Implementations. In *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*, pages 79–98, 2004.
- [8] S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and Systems Management*, 13(2), 2005.

Appendix A

In this Appendix we show the results of the comparison between our Cyclon and ADH implementations against the original ones. For Cyclon we compared our implementation with the one provided by the authors of [8]. The tests were conducted as follows: starting from an initial regular random graph of 1000 nodes we measured the in-degree distri-

bution after 100 shuffle cycles without overlapping (cache size 20 and shuffle length $l = 5$). The tests (Figures 3) always returned the same distribution of nodes' degree meaning that our implementation is consistent with the protocol's original specification. For ADH we lack an original implementation thus we just compared curves returned by our implementation with those provided in [2]. More specifically, Figure 4 shows how the mean time needed from the overlay network to experience the first partitioning varies with the view size. The same plot is shown as Figure 6.1 in [2]. While the reported values are completely different (mainly due to the different scenarios used by the authors of [3, 2] to test their OMP), our implementation shows the same overall behaviour.

Evaluation of Unstructured Overlay Maintenance Protocols under Churn

R. Baldoni S. Bonomi A. Rippa L. Querzoni S. Tucci Piergiovanni A. Virgillito
Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”
Via Salaria 113, 00198 Roma, Italia

Abstract

An overlay network is formed on top of – and generally independently from – the underlying physical computer network, by the peers (nodes) of a P2P system. The dynamics of peers is taken into account by devising appropriate overlay maintenance protocols that are able to join and leave peers from the overlay. Due to the need for scaling in the number of nodes, overlay maintenance protocols have been simulated only in environments showing a very restricted behavior with respect to the possible concurrent and interleaved execution of join/leave operations.

In this paper we compare two overlay maintenance protocols well suited to unstructured P2P systems, namely SCAMP and Cyclon, in an event-based simulation setting including concurrent and interleaved join and leave operations as well as variable message transfer delay. This simulation setting allows to point out surprising results for both protocols. In particular, under a continuous and concurrent replacement of nodes, permanent partitioning of the overlay arises after a very small number of join/leave operations.

1. Introduction

P2P systems are at present a widespread technology as well as a hot research topic. A P2P system is a highly dynamic distributed system in which nodes perpetually join and leave. For these characteristics, a P2P system can reach a potentially infinitely wide scale with a transient population of nodes.

Overlay maintenance is a fundamental problem in peer-to-peer (P2P) systems. An overlay is a logical network built on top of – and generally independently from – the underlying physical computer network, by the peers (nodes) of the P2P system. Any overlay should exhibit a topology able to support a P2P application in an efficient and scalable manner maintaining a satisfactory level of reliability. Unstructured overlay networks have emerged as a viable solution to settle such issues [3, 4, 5, 8, 10] in order to effectively support large scale dissemination and flooding-

based content searching. An unstructured overlay shows good global properties like connectivity (for reliability), and low-diameter and constant-degree (for scalability) without relying on a deterministic topology. To cope with the inherent P2P dynamics, however, a so-called *overlay maintenance protocol* (OMP) is needed. The main goal of any OMP is properly arranging the overlay to keep as much as possible the desired global properties of the overlay over the time, despite the *continuous and interleaved process of arrival/departure of nodes*, i.e., *churn*. Amongst the most popular OMPs that do not use a central server we cite [3, 4, 5, 10].

All the above cited works include an experimental evaluation of the protocols in which basic topological properties of the overlay are evaluated, such as resilience to failures (reliability) and distribution of node degree (scalability). However, at the best of our knowledge, the conditions under which experiments are made only consider a limited amount of possible dynamic behaviors. For example one of such typical experimental scenarios (as considered in [5, 7, 10]) is divided in two phases where, firstly, all nodes in the system join and, successively, a portion of nodes leaves the system simultaneously. Moreover, each phase is divided in rounds and each node executes at most one join/leave operation atomically in a round, i.e. the execution of two operations in the same round cannot interleave. This type of experiments is intended to simplify the computation of the simulation in order to scale the simulation itself in the number of processes (usually these simulations reach 100.000 nodes) and then to evaluate, for example, the portion of nodes that can simultaneously leave the overlay without creating a partition in the overlay topology.

We believe that a further step is required to analyze the characteristics of OMPs in scenarios where more dynamic behaviors are admitted that actually mimic the possible dynamics occurring in realistic P2P environments. The challenge is to check if reliability and scalability properties are still preserved in this more severe setting. For example, a high interleaving between joins and leaves could *permanently* spoil the overlay connectivity, leading to a higher probability of node isolation and partitioning. Moreover,

unpredictable message delays (typical in a wide-area network), can provoke interleaving between messages sent in different rounds of a protocol, causing inconsistency in views at nodes that, again, has a negative effect on the overlay properties.

In this paper we present the results of a first attempt in evaluating OMPs behavior under churn. In particular, the focus is in evaluating the overall robustness of the protocols over the time despite *continuous overlay node replacements* i.e., new nodes join the system while others leave. We chose two particular protocols, namely SCAMP [5] and CYCLON [10] as representatives of two different approaches to overlay maintenance, respectively *reactive maintenance*, where the protocol undertakes actions in rearranging the overlay only upon arrival of nodes, and *proactive maintenance*, where each node continuously gossips membership information (i.e., its view) among its logical neighbors. Proactive maintenance protocols allow a better resilience to high churn in terms of concurrent join/leave operations per time unit at the price of a persistent activity of nodes, inducing a constant overhead on the network. Reactive maintenance protocols are more suited to environments showing a “moderate” number of concurrent operations per time unit, where they eliminate the gossip overhead in period of inactivity.

Protocols were implemented in the same simulation environment, namely Peersim [1]. Differently from other works using the same tool [10, 7], where simulations were performed following a round-based approach, here we use an event-based approach, in order to introduce aspects such as join/leave interleaving and unpredictable message delays. All such elements induce a high degree of concurrency in a run of a simulation, that was not present in the round-based simulations. This simulation shift brings to reduce the magnitude of the P2P systems to be analyzed (order of thousands of nodes) due to the enormous resource consumption. Nevertheless, P2P systems formed by thousands of nodes are big enough to point out the main characteristic of each OMP protocol under churn.

Starting from an ideal P2P overlay network, the results of the simulations show the difficulty of the tested protocols to face continuous node replacement. Permanent partitioning starts to occur when a low percentage of nodes forming the initial P2P network has been replaced by new joining nodes. This result is surprising when compared to churn-free (i.e., no join/leave operation interleave) simulations of the same protocols that showed partitioning only when a high percentage of nodes left concurrently the system. Though as expected the proactive approach of Cyclon results more suitable to resist to churn than SCAMP, its ability to recover full connectivity strictly depends on the frequency of gossiping. Concerning SCAMP, the resistance to churn depends on the percentage of number of initial nodes

that remain in the system. If this percentage is below than 90%, the process of churn tends to disaggregate the overlay topology quite early leaving connected only a very small fraction of the nodes in the system.

We believe that this work, though not intended to represent a comprehensive simulation study, clearly indicates that the impact of churn in OMPs deserves further study.

2. Protocols Description

The common characteristic of all OMPs is that each node maintains a limited number of links to other nodes in the system. We call this set of links the *view* of the node. The views should be such that the graph, resulting by interpreting links in the view as arcs and nodes as vertexes, is connected. OMPs differentiate among themselves with respect to the techniques they use for building and maintaining the views. We consider decentralized OMPs in which such protocols do not require a central coordination. In this Section we describe in detail the two protocols that are subject of our study.

2.1. SCAMP

SCAMP [5] is a gossip-based protocol whose main innovative feature is that the size of the view is adaptive w.r.t. a-priori unknown size of the whole system. More precisely, view size in SCAMP is logarithmic of the whole system size. The protocol consists of mechanisms for nodes to join and leave, and to recover from isolation. The following is a brief description of these mechanisms.

Data Structures. Each node maintains two lists, a *PartialView* of nodes it sends messages to, and an *InView* of nodes that it receives messages from, namely nodes that contain its node-id in their partial views.

Join Algorithm. New nodes join the overlay by sending a join request to an arbitrary member, called a *contact*. They start with a *PartialView* consisting of just their contact. When a node receives a new join request, it forwards the new node-id to all members of its own *PartialView*. It also creates c additional copies of the new join request (c is a design parameter that determines the proportion of failures tolerated) and forwards them to randomly chosen nodes in its *PartialView*. When a node receives a forwarded join request, provided the subscription is not already present in its *PartialView*, it integrates the new node in its *PartialView* with a probability $p = 1/(1 + \text{sizeofPartialView}_n)$. If it decides not to keep the new node, it forwards the join request to a node randomly chosen from its *PartialView*. If a node i decides to keep the join request of node j , it places the id of node j in its *PartialView*. It also sends a message to node j telling it to keep the node-id of i in its *InView*.

Leave Algorithm. The leaving node orders the id's in its *PartialView* as $i(1), i(2), \dots, i(l)$ and the id's in *In-*

View as $j(1), j(2), \dots, j(l)$. The leaving node will inform nodes $j(1), j(2), \dots, j(l - c - 1)$ to replace its id with $i(1), i(2), \dots, i(l - c - 1)$ respectively (wrapping around if $(l - c - 1) > l$). It will inform nodes $j(l - c), \dots, j(l)$ to remove it from their lists without replacing it by any id.

Recovery from isolation. A node becomes isolated when all nodes containing its identifier in their PartialViews have either failed or left. In order to reconnect such nodes, a heartbeat mechanism is used. Each node periodically sends heartbeat messages to the nodes in its PartialView. A node that has not received any heartbeat message in a long time re-joins through an arbitrary node in its PartialView.

2.2. Cyclon

Cyclon [10] follows a proactive approach, where nodes perform a continuous periodical gossiping activity with their neighbors in the overlay. The periodical gossiping phase (named “shuffle cycle”) has the aim of randomly mixing the views between neighbor nodes. Clearly, joins are managed in a reactive manner, while voluntary departures of nodes are handled like failures (no leave algorithm is provided). A failure detection mechanism is provided in order to clean views from failed nodes.

Data Structures. Each node maintains only a single view of nodes it can gossip with (i.e., it corresponds to SCAMP’s PartialView). The size of the view is fixed and it can be set arbitrarily. Each node in the view is associated to a local age, indicating the number of shuffle cycles during which the node was present in the view.

Join Algorithm. A node A joins by choosing one node (*contact*) at random among those already present in the network. The contact starts then a set of independent random walks from the contacted node. The number of random walks is equal to the view size, while the number of steps per each random walk is a parameter of the algorithm. When each random walk terminates, the last visited node, say B , adds A to its view by replacing one node, say C , which is added to A ’s view using an empty slot.

Shuffle Algorithm. The shuffle algorithm is executed periodically at each node. A shuffle cycle is composed of three phases. In the first phase a node A , after increasing the age of all the nodes in its view, chooses its shuffle target, B , as the one with higher age among those in its view. Then, A sends to B a shuffle message containing $l - 1$ nodes randomly chosen in A ’s view, plus A itself. In the second phase, B , once received the shuffle message from A , replaces $l - 1$ nodes in its view (chosen at random) with the l nodes received from A and send them back to A . In the final phase A replaces the nodes previously sent to B with those received from it. Overall, the result of one shuffle cycle is an exchange of l links between A and B . The link initially present from A to B is also reversed after the shuffle.

Handling Concurrency. In the specifications given in [10],

no action was defined in the scenario of two (or more) *concurrent* shuffle cycles, e.g. when a node A , during a shuffle cycle in progress with B , is selected as a target node by receiving a shuffle message from C . If concurrency is considered, the nodes sent by A to B can be modified by the concurrent shuffle involving A and C . In our implementation, we extend the original specification in order to address this situation: in case nodes to be replaced by A are no longer in its cache, it replaces some nodes chosen at random.

3. Simulation Study

In this Section we present the details of our simulation study. Results of the two protocols are presented separately¹. We point out that this work is not intended to be a comparison between the two protocols, since they were designed for different purposes². The simulations aims only at showing the behavior of these different protocols under conditions of churn and concurrency.

3.1. Experimental Setting

The simulation study was carried out by developing the two algorithms in Peersim [1]. The event-driven mode of Peersim was used for both protocols. Event-driven simulations in Peersim are based on a logical clock. At each time unit t of the clock one or more events can be scheduled. The scheduled events are: *join invocation*, *leave invocation*, *send* and *receive* of messages.

Differently from the cycle-driven mode of Peersim, the event-driven mode allows to introduce concurrency. In particular, (i) the not synchronized execution of joins, leaves and shuffle cycles, and (ii) the random delay between the send and receive of a message, allows joins, leaves and shuffle cycles to take a variable amount of time to execute, augmenting the possibility of overlapping.

Simulations for both protocols were carried out as follows. A run of a protocol is divided into three periods: *creation*, *churn* and *stability*. During the creation period, nodes join until reaching a given value N . Neither leaves nor overlapping of joins occur along this phase. During the churn period, nodes continuously join and leave the network at a given *churn rate* C , i.e. at each unit of time, C nodes invoke the join and C nodes invoke the leave. The churn period ends when 3000 joins and 3000 leaves have occurred. Thus, the churn period duration varies in function of the churn rate and, at the end of this period, the total number

¹Both protocols implementations were validated comparing to the ones presented in [10] and [5] respectively. This comparison is shown in the Appendix A.

²SCAMP was originally targeted at the construction of overlays for large-scale information dissemination, for which the reactive nature of the protocol is more appropriate, while Cyclon is in general suited for applications requiring a constant sampling of nodes in the network, e.g. searching, monitoring, etc.

of nodes in the overlay is still N . N was set to 1000 in all experiments³. Message delay varies uniformly at random between 1 and 10 time units. 10 independent runs were made for each experiment.

The metrics we focus on are (i) the average percentage of reached nodes (R) and (ii) the overlay clustering at the end of the stability period.

Evaluating average reachability. This metric is defined as the average number of nodes that can be reached from any node in the overlay, with respect to the total number of nodes. This metric is obviously related to the connectivity of the overlay graph, as any value lower than 100% indicates that at least one node cannot be reached by at least one other node. We evaluated the effect of the variation of the churn rate C ($C=2,4,8,16,32,64$) on R along the time (each point in these experiments is taken every 100 join and 100 leave invocations). In order to facilitate the comparison between experiments resulting by simulating different churn rates, the churn period has been made equal for any churn rate by expressing the execution time as a normalized time ($\tau = \frac{t * C}{3000} * 100$). Thus, a same value of normalized time corresponds to a same number of invoked joins and leaves for any churn rate. Both protocols have been evaluated with leaving nodes chosen uniformly at random in all experiments. Experimental results show that at the end of the churn period the set of initial 1000 nodes are almost completely replaced (in average, the 4% of the initial nodes remains during the entire simulation). For SCAMP we have also evaluated the impact on R of different policies in the choice of the leaving nodes while for Cyclon the impact on R of different shuffle frequencies.

Overlay clustering. In order to highlight the type of overlay connectivity when R is lower than 100%, we also show the clustering of the overlay at the end of the stability period: the percentage of nodes forming the maximum connected component (main cluster), the percentage of isolated nodes and the percentage of nodes forming clusters with dimension less than the 6% of the overlay⁴.

3.2. Evaluation of SCAMP

The results of experiments for SCAMP are presented in figures 1(a), 1(c) and 1(e). For SCAMP we chose a heartbeat period equal to 50 time units.

In the first experiment (Figure 1(a)) we tested the average reachability R along the time under churn. The plot clearly illustrates the dependence of R from C , showing how churn can permanently disrupt the overlay connectivity. For churn rates higher than 4, at the end of each run, R is close to 0%, meaning that the topology is entirely fragmented into small-sized partitions and many nodes become permanently

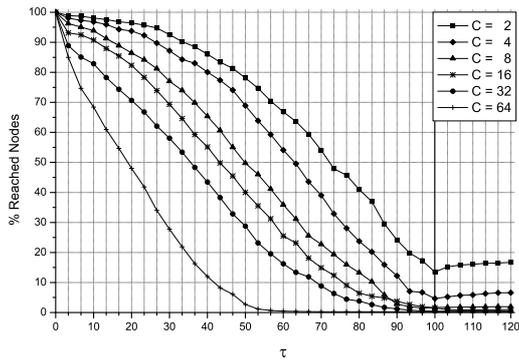
isolated as showed in Figure 1(c). We remark the great difference with the results showed in [5], in which R is equal to 99% even after 50% of the nodes have been removed from the network. In our scenario, R starts to deviate from 99% when only 10% of nodes have been replaced (for $C = 2$ after this substitution, i.e. for $\tau = 10/3$, $R = 98,9\%$). The main reason behind this behavior is the poor connectivity of nodes replacing the old ones during churn period. Initially, the overlay is formed by 1000 well-connected nodes. After the replacement of the 10% of nodes, for $C = 2$, we have a degradation in connectivity since the new 10% is poorly connected with respect the replaced 10%. The reason lies in the fact that the old 10% was obtained in an ideal manner during the creation period, while the new 10% has been added to an overlay suffering from node departures and simultaneous joins (nodes joining concurrently are connected among them through the initial overlay disrupted by the deletion of some nodes). During the churn period connectivity keeps degrading with the progressive replacement of nodes in the overlay. While R is greater then 80%, the more the velocity of replacement the worst the connectivity shown by the replacing part of the overlay, e.g. for $C = 2$ after the replacement of 10% we have $R = 98,9\%$ while with a higher churn rate $C = 4$ after the replacement of a same 10% we have $R = 98\%$. However, for lower values of R , the slope of different curves become almost the same, pointing out a sub-linear degradation of R with respect to C and the dominating effect of the quantity of replaced nodes versus the velocity of their replacement. Interestingly, after the churn stops ($\tau = 100$) there is a small raise in R , for the churn rates lower than 8, witnessing the effect of the heartbeat mechanism during the stability period.

It is clear that, under these conditions, it becomes critical for the protocol the presence of a well-connected cluster of nodes not subject to replacement. For testing this effect, in the second experiment we consider a variable percentage of nodes to be “permanent”, i.e. nodes joining during the creation period and never leaving the overlay, with a fixed churn rate equal to $C = 2$. Figure 1(e) shows the results of the experiment when changing the percentage of permanent nodes. Values chosen were 0%, 10%, 50% and 90%. In the “Random” curve, nodes leaving the overlay were chosen at random, as in the previous experiment⁵. The plot shows the positive effect of the permanent nodes over R . The percentage of reached nodes during the stability period is always higher than the number of permanent nodes, meaning that the presence of a fixed connected cluster facilitates new joining nodes to remain connected to the main cluster.

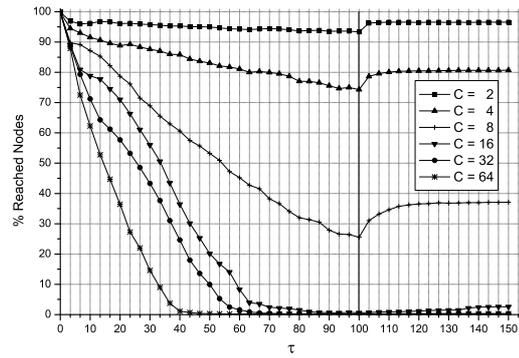
³In Appendix B is shown that changing the initial size with the same ratio between N and C brings to obtain the same experimental results.

⁴As we will see later, no cluster of size higher than 6% is ever created.

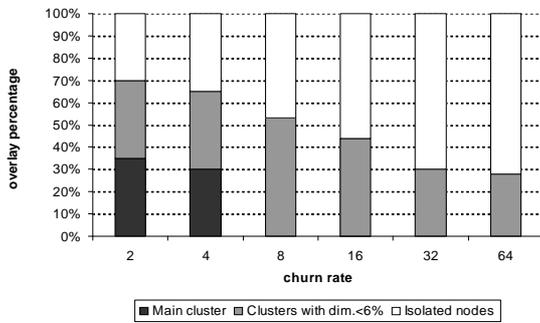
⁵Random performs better than 0% because some permanent nodes (in average the 4%) are present



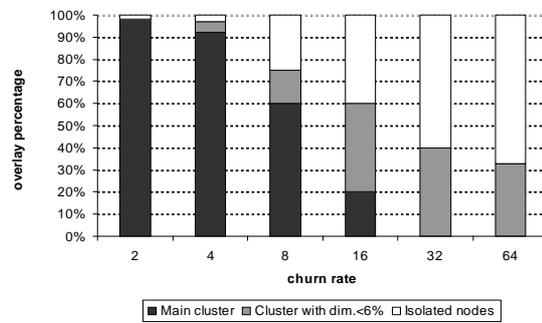
(a) SCAMP: Variation of R along time with different churn rates



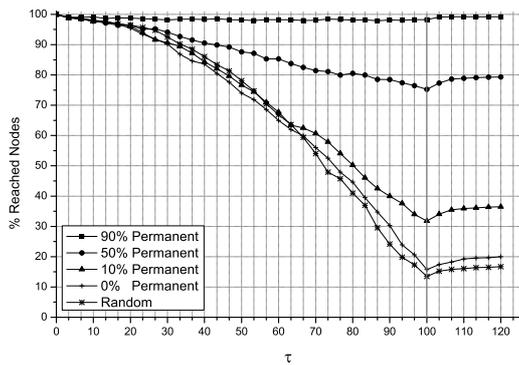
(b) Cyclon: Variation of R along time with different churn rates



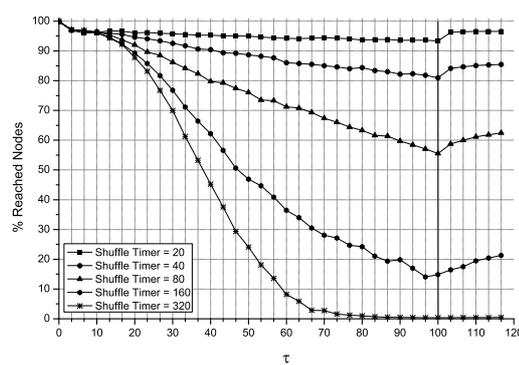
(c) SCAMP: Overlay clustering at the end of the stability period with different churn rates



(d) Cyclon: Overlay clustering at the end of the stability period with different churn rates



(e) SCAMP: Variation of R along time with different percentages of permanent nodes with fixed churn rate $C = 2$



(f) Cyclon: Variation of R along time with different shuffle rates with fixed churn rate $C = 2$

Figure 1. Experimental Results

3.3. Evaluation of Cyclon

All experiments with Cyclon use a view size set to 7, being the logarithm of the system size. The shuffle length l is 2, while the length of random walks in the join is 5.

In the first experiment (Figure 1(b)), we tested the effect of the variation of the churn rate C on R while Figure 1(d) shows the clustering of the overlay in all cases. The shuffle period is set to 20 time units. Again, a severe churn rate permanently disrupts the overlay connectivity, with nodes getting isolated from the largest partition. As a comparison with the results presented in [10], where R starts to decrease when 75% of nodes are removed, in our experiments R is lower than 100% starting from the first point (10% of substituted nodes at $\tau = 10/3$). There is an important difference with SCAMP: the churn rate affects more significantly the trend of R : the overall number of replacements is unimportant (with $C = 2$, R remains almost 100% despite the number of overall replacements), the dominating effect is the velocity of the replacement since it impacts on the efficiency of the shuffle mechanism: a slower replacement implies a higher number of shuffle cycles.

In Figure 1(f), we test the effect of varying the shuffling period. The churn rate is fixed to $C = 2$ and the shuffling timer varies from 20 to 320 time units. As expected, R decreases faster with higher shuffling periods. Also the convergence in the stability period is slower. Finally, it is interesting comparing results in Figures 1(b) and 1(f) focusing on those curves where the number of operations between two shuffle periods is the same. For instance, let us observe the curve for $C = 8$ in Figure 1(b) (Shuffle Timer=20 and $C = 8$) and the curve for Shuffle Timer=80 in Figure 1(f) (Shuffle Timer = 80 and $C = 2$): in both experiments there are approximately 160 join and 160 leave invocations before that a node shuffles. The fact that R is always higher in the first test, indicates that the velocity of replacement dominates over the number of replacements, making the shuffle less effective though it is performed more frequently.

4. Related Work

Different distributed OMPs supporting gossip-based dissemination have been proposed [4, 5, 3, 10, 2]. These protocols provide each node with a small local view of the overlay membership at each node and membership information spreads in an epidemic style [6]. However, [4, 5] do not take into account the issue of the overlay changing rate explicitly. In [3] the authors express, through an analytical study, the time expected for the overlay to partition as a function of (i) the overlay size, (ii) the local view size and (ii) the overlay changing rate (called churn rate). The local view size needs to be larger than the churn rate for the expected time until partitioning to be exponential in the square of the local view size. The protocol proposed, however, has not

been evaluated through an event-based simulation study, i.e. under concurrency and random message delays.

For completeness we also cite [2] since it is the algorithm that first introduces the main features of Cyclon: shuffles cycles and random walks. Cyclon is an improvement w.r.t. to [2] obtained by using the aging mechanism.

This work extends and deepens the first results presented in [9] in which we began to evaluate the SCAMP behavior under churn with a very small overlay (only 100 nodes).

5. Concluding Remarks

The aim of the paper has been to test the robustness of the overlays obtained from SCAMP and Cyclon protocols with the precise intent to stress each protocol under severe churn situations, in order to determine their breakdown behavior.

Other aspects need further investigation. For example, in our experiments we assumed all nodes initially joining the system are not “disturbed” by concurrent leaves. This brings to the construction of an ideal initial overlay network. Now the problem is how one can set up a network with thousands of nodes in that way. A more realistic model should take this into account to see the effect of operation interleaving starting at a very early stage when the size of the P2P system is in the order of a more realistic tens of nodes.

References

- [1] *Peersim*, <http://peersim.sourceforge.net/>.
- [2] D. Rubenstein A. Stavrou and S. Sahu, *A Lightweight, Robust P2P System to Handle Flash Crowds*, IEEE Journal on Selected Areas in Communications **22** (2004).
- [3] A. Allavena, A. Demers, and J. E. Hopcroft, *Correctness of a Gossip Based Membership Protocol*, Proceedings of the 24th ACM annual symposium on Principles of Distributed Computing (PODC05), 2005, pp. 292–301.
- [4] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, *Lightweight Probabilistic Broadcast*, ACM Transactions on Computer Systems **21** (2003), no. 4, 341–374.
- [5] A. Ganesh, A. Kermarrec, and L. Massoulié, *Peer-to-Peer Membership Management for Gossip-based Protocols*, IEEE Transactions on Computers **52** (2003), no. 2, 139–149.
- [6] R. A. Golding and K. Taylor, *Group Membership in the Epidemic Style*, Tech. Report UCSC-CRL-92-13, 1992.
- [7] M. Jelasity, R. Guerraoui, A. Kermarrec, and M. van Steen, *The peer sampling service: Experimental evaluation of unstructured gossip-based implementation*, Proceedings of Middleware 2004, 2004.
- [8] G. Pandurangan, P. Raghavan, and E. Upfal, *Building Low-Diameter p2p Networks*, IEEE Symposium on Foundations of Computer Science (FOCS01), 2001, pp. 492–499.
- [9] R. Baldoni, A. Noor Mian, S. Scipioni, and S. Tucci Piergiovanni, *Churn Resilience of Peer-to-Peer Group Membership: a Performance Analysis*, In Proceedings of the International Workshop on Distributed Computing, December 2005.
- [10] S. Voulgaris, D. Gavdia, and M. van Steen, *CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays*, Journal of Network and Systems Management **13** (2005), no. 2.

Appendix A

In this Appendix we show the results of the comparison between our SCAMP and Cyclon implementations against the original ones, respectively GKM implementation [5] and VGvS implementation [10].

We made the comparison between the two implementations of Cyclon, using the original one provided in the Peersim library [1]. We simulated the following: starting from an initial regular random graph of 1000 nodes we measure the in-degree distribution (in-degree for a node is the number of nodes that it receives messages from) after 100 shuffle cycles without overlapping shuffle cycles (cache size 20 and shuffle length $l=5$).

For SCAMP, as we do not have the original implementation available, we made our simulations with the same parameters under which original results were obtained (degree distribution of 100000 initial nodes after the removal of the 50% of nodes). Our experiments (Figures 2, 3) returned the same distribution of nodes degree as the original experiments meaning that our implementation is consistent with the protocols' original specifications.

Appendix B

In this Appendix we show the results of R obtained starting from different N but maintaining the same ratio between C and N . We compared the curve obtained with 1000 and $C=2$, with the results obtained doubling the parameters ($N=2000$ and $C=4$) and halving the parameters ($N=500$, $C=1$). Figures 4 and 5 show how, both for Cyclon and SCAMP, maintaining unaltered the ratio between N and C brings to the same impact on R .

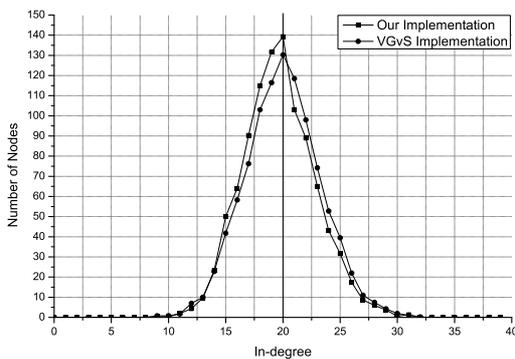


Figure 2. Comparison between degree distribution of our Cyclon implementation and the original one

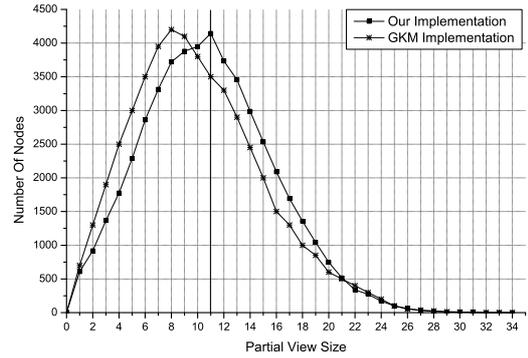


Figure 3. Comparison between degree distribution of our SCAMP implementation and the original one

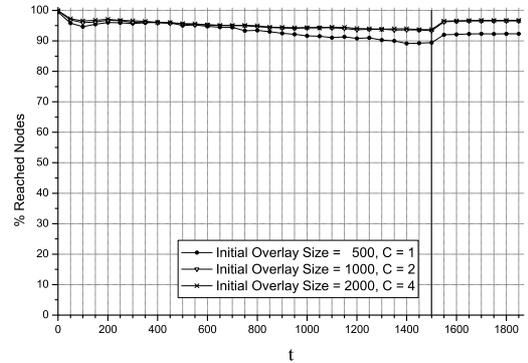


Figure 4. Cyclon: variation of R for different N and the same ratio N/C

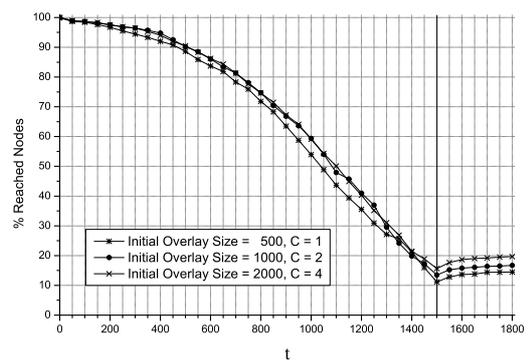


Figure 5. SCAMP: variation of R for different N and the same ratio N/C

Weakly-Persistent Causal Objects In Dynamic Distributed Systems

R. Baldoni, A. Milani, S. Tucci Piergiovanni

Università di Roma *La Sapienza*

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"

Via Salaria 113, Roma, Italia

{baldoni,milani,tucci}@dis.uniroma1.it

M. Malek

Humboldt Universität zu Berlin

Unter den Linden 6, 10099 Berlin, Germany

malek@informatik.hu-berlin.de

Abstract

In a large scale system with a potentially infinite number of clients accessing a read/write shared object, persistency of values written (i.e., a value written is always available if no more write operations are issued) could be easily guaranteed by implementing the object through $2f + 1$ servers where up to f of them may crash. In this paper we explore a different scenario, where processes implementing the object may fail, join or leave at any time of the computation. The paper on one hand points out that in such systems, objects are naturally not persistent due to the continuous arrival and departures of processes implementing the object but, on the other hand, a so-called weak persistency could be incorporated, i.e., the property of guaranteeing persistency when a system becomes quiescent (arrivals and departures subsided).

Finally, an implementation of a weakly-persistent causal object along with its correctness proof is given.

1. Introduction

This paper focuses on the problem of implementing shared objects over an asynchronous message passing system characterized by (i) infinitely many processes and (ii) high dynamics: processes may join or leave the computation at any time. This dynamic distributed system model abstracts continuously running systems like peer-to-peer systems.

In order to implement objects in this environment, we adopt the client/server paradigm and the related failure model proposed in [8]. More specifically, clients coordinate

the access to the object through servers and no communication among clients is assumed. Then, the set of clients may be infinitely large. The object is implemented by a fixed set of virtual servers. At any time a process incarnates a virtual server. Upon the crash (or the leave) of such a process, a new process replaces the old one in incarnating the virtual server and the old state of the failed process is completely lost. This demands a system model in which processes crashes are associated with *memory losses*. To model possible infinite alternation of peers, incarnating a virtual server, these losses can be an infinite large number. This dynamic system model nicely captures for example the basic behavior of structured P2P systems, [?].

Motivation. As the object can be concurrently accessed by read and write operations, clients must be provided with a consistency criterion that defines object semantics. Atomic consistency is recognized to be the most useful since it provides the client processes with the illusion that they access the memory one at a time [17]. However, our system model does not allow atomic objects implementations. In particular, in a crash prone system, atomicity may be guaranteed provided that object state persistency is ensured through crashes [13]. Due to the assumption of an *arbitrarily large number of memory losses*, object state persistency may not be ensured. Thus, we consider a weaker consistency criterion, namely causal consistency [3], which is proved to be implementable in such a system as it can tolerate an infinite number of memory losses.

Contribution. The paper provides a protocol implementing a causal object in a continually running and dynamic system affected by an arbitrarily large number of memory losses.

A causal object ensures that values returned by read op-

erations are consistent with the causality order relation. In particular, if the write operation of a value a , namely $w(x)a$, causally precedes the one of a value b , namely $w(x)b$, every client process that reads both values, has to read a and then b . Let us remember that $w(x)a$ causally precedes $w(x)b$ if i) both writes are issued by the same client process and $w(x)a$ is issued before $w(x)b$, or ii) the client issuing $w(x)b$ reads the value written by $w(x)a$ before issuing $w(x)b$ or iii) because of transitivity.

After memory losses, new clients could miss completely the computation already done due to lack of persistency of written values. Under these circumstances, a protocol implementing a causal object should not block any operation (new clients never block waiting for lost values) while at the same time guaranteeing a safe behavior w.r.t. causal consistency. However, let us remark that in this case, causal consistency can be ensured by a trivial protocol that, for every read operations, returns the initial value of the object.

To cope with this problem, the paper introduces a property, called *weak persistency* that, on one hand, is strong enough to rule out trivial implementations and on the other hand, is weak enough to be implemented in our system model. More specifically, in periods in which the system is *quiescent* (each process incarnates a virtual server **forever**) computation as perceived by a client continually makes progress, that is clients are able to read the most recent values.

We propose a protocol, along with its correctness proof, implementing a so called weak-persistent causal object. The protocol enjoys the desirable property of maintaining causal consistency *all the time* regardless of periods affected by high dynamics and of leveraging *quiescent periods* to bring forward a computation perceived in the same way by all clients joining the system along the time.

Road-Map The paper is structured into six sections. Section 2 describes the object model and the consistency model. Section 3 specifies the system model and our definition of weak persistency. In Section 4, we give the implementation of a weakly-persistent causal object along with its correctness proofs. In Section 6, we consider the related works and finally we present conclusions in Section 7.

2. Object Model

Client processes interact via a shared object x through read and write operations. A write operation aims at storing a new value in object x , while a read is supposed to return the value stored in x . Object x is initialized to \perp . Each client process is univocally identified by a positive integer, i.e. c_i will denote the client process whose identity is i . Thus, formally: we denote as $w_i(x)v$ a write operation invoked by a client process c_i to store a value v in x and as

$r_i(x)v$ a read operation invoked by a client process c_i and that returns to c_i the value v stored in x . We assume that each write operation is univocally identifiable. In detail, a write may be identified by the value written and the process identifier provided that the client does not write more than once the same value, otherwise, it is sufficient to additionally consider a sequence number.

As the object can be concurrently accessed (by read and write operations), clients must be provided with a consistency criterion that precisely defines the semantics of the shared object, that is the value each read operation has to return. A consistency criterion defines correctness in terms of histories.

History properties Since clients are sequential processes, each client c_i generates a sequence of operations called *local history* and denoted h_i . A *history* H , is the union of all local histories, one for each client process.

Causality order relation Given a history H , let o_1 and o_2 be two operations in H , $o_1 \mapsto_{co} o_2$ if and only if one of the following cases holds:

- $\exists c_i$ s.t. o_1 precedes o_2 in c_i program order,
- $\exists c_i, c_j$ s.t. $o_1 = w_i(x)v$ and $o_2 = r_j(x)v$ (read-from order),
- $\exists o_3 \in H$ s.t. $o_1 \mapsto_{co} o_3$ and $o_3 \mapsto_{co} o_2$ (transitive closure).

Two operations o_1 and o_2 are *concurrent* w.r.t. \mapsto_{co} , denoted $o_1 \parallel_{co} o_2$, if and only if $\neg(o_1 \mapsto_{co} o_2)$ and $\neg(o_2 \mapsto_{co} o_1)$.

Causal Consistent Object A read/write causal consistent shared object x is characterized by the following properties:

Definition 1 (Legality.). *Given a history H , if it exists a read operation $r(x)v$ belonging to H , then i) there must exist a write operation $w(x)v \in H$ such that $w(x)v \mapsto_{co} r(x)v$ and ii) there must not exist a write operation $w(x)v' \in H$ such that $w(x)v \mapsto_{co} w(x)v'$ and $w(x)v' \mapsto_{co} r(x)v$.*

Definition 2 (Causal Ordering.). *Given a history H , let $w(x)v$ and $w(x)v'$ be two write operations belonging to H and such that $w(x)v \mapsto_{co} w(x)v'$. If a client process c_i reads both values written by such write operations, namely v and v' , then c_i first reads v and then v' .*

3. System Model

We consider the *infinite arrival* model proposed in [1]: the system consists of possibly infinitely many processes,

runs can have infinitely many processes, but in each time interval only finitely many processes take steps. The system is asynchronous, that is there is no bound on the relative process speeds, however, the time taken by each process to execute a computational step is finite. Moreover, message transfer delay is finite but unpredictable. As depicted in Figure 1, components of the system are logically separated in: client processes, object entities and object manager processes.

Object x is implemented by a finite number n of virtual servers, also called object entities $\{x_1, x_2, \dots, x_n\}$. Each object entity is characterized by an univocal virtual identifier and a state. In particular, x_j denotes the j -th object entity and its state is its current value.

Each object entity x_i is implemented by an object manager process which is in charge of the actual execution of read/write operations invoked by client processes. An object manager process is identified by the identity of the object entity it is in charge of. Since at each time, each object entity is incarnated by a single object manager process, sometimes we denote as x_i both the object entity and the corresponding object manager process.

Client processes communicate with object manager processes exchanging messages over *fair-loss point-to-point channels* [21]. There is no communication among object manager processes.

Failure Model A process (client or object manager) may crash, that is, it halts prematurely. A crashed process does not recover. This means that from a practical point of view, a process that crashes, can re-enter the system with a new identity. A process that does not crash is correct otherwise it is faulty.

We treat the deliberate leave of an object manager as a crash. If an object manager leaves the system, deliberately or by crashing, if a new object manager will replace that previous one it will assume the same virtual identity. As an example, in Figure 1, the process i crashes and it is replaced by process k . Moreover, the new object manager process is not able to retrieve any state the crashed process passed through during its execution. We assume that each time an object manager process leaves the system, there exists a new one that *may* replaces the previous one. For what said, each object entity x_i is characterized by a sequence of object managers, denoted \hat{x}_i .

Let us remark that the mapping between object entities and object manager processes can be realized through well-known technologies such as Domain Name Server (DNS), Distributed Hash Table (DHT) etc. This technologies include mechanisms providing a good support for maintaining a stable set of server processes. Thanks to the possibility of having concurrent joins and leaves, the system model

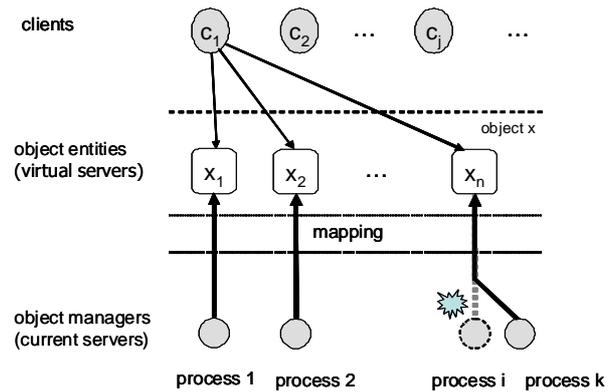


Figure 1. An architecture of a weakly consistent causal object.

is well-suited to represent an object implementation on the top of a structured peer-to-peer system.

Weak persistency and Weakly-Persistent Causal Object

Intuitively, values are persistent if, in absence of new write operations, subsequent reads may return the last value written. But, for a causal object the concept of last value written is not univocally determined, i.e. due to causally concurrent write operations different clients may perceive as last written value a different value.

Moreover, due to our failure model we propose a weak form of persistency, i.e. persistency is only guaranteed for values written in quiescent periods of the system, that is when the set of processes incarnating object managers does not leave the system deliberately or by crashing.

Formally,

Definition 3 (Weak Persistency). *If a value v is written infinitely many times, then a client process that reads infinitely many times, eventually reads v , a concurrently written value or a subsequent one w.r.t. \mapsto_{co} .*

This property abstracts the fact that the object may suffer a finite (and unknown) number of memory losses when the set of processes incarnating the object frequently changes but never loses memory during periods in which this set does not change for a time long enough. According to this, we define what we call *weakly-persistent causal object*. Formally:

Definition 4 (Weakly-Persistent Causal Consistent Object). *A weakly-persistent causal consistent object is a causal consistent object satisfying the property of weak persistency.*

4. Weakly-Persistent Causal Consistent Object Implementation

Client processes invoke operations by sending request messages to the set of object entities. Since each object entity x_i is incarnated by an object manager process that may change during time, we assume the existence of an underline routing system that is able to route request messages to the object manager process that currently incarnates x_i . When an object entity receives a request of a client process c_i , it processes that request and then it sends the corresponding response to c_i . A correct implementation has to satisfy the following properties:

Definition 5 (Termination). *If a correct client process c_i invokes an operation, then c_i eventually returns from the invocation.*

Definition 6 (Validity). *If a read operation invoked by a client process c_i returns a value v , then there exists a client process c_j that invoked the write of v .*

Finally, we make the following assumption:

Assumption 1. *There are $\lceil 2n + 1/3 \rceil$ object entities x_i , whose corresponding \hat{x}_i is finite.*

4.1. Data Structures

Each client process c_i has to manage: 1) `ack[1..n]`: a vector of boolean, one for each object entity. Each entry is initially set to false. It is used to track when $f = \lceil 2n/3 \rceil$ object entities have answered to a read request made by c_i . $ack[k] = true$ means that c_i has received from x_k a response to its current read request; 2) `ack`: an integer initially set to 0. It stores the number of ack received by c_i from object entities in order to track when an ack is received by f object entities. Each object manager x_i has to manage a variable `last`, to track the client that invoked the write operation corresponding to the last value stored at x_i . This information is used to check causal consistency.

Moreover, in order to guarantee causal consistency, processes in the system, both clients and object managers, have to manage a timestamping system to implement a plausible clock t [5]. The plausible clock system we propose is an adaptation of *R-Entries vector clock system (REV)* proposed by Ahamad et al. in [5]. Each process stores a vector of integers of fixed size n , initially set to $[0, \dots, 0]$. This vector is denoted $t_i[1..n]$ for a client process c_i and $tx_i[1..n]$ for an object manager x_i . Each client process c_i is associated to the $i \text{ modulo } n$ entry of the plausible clock t . According to this and due to the fact that the number of client processes in the system may be more than n at a given point in time, several clients may share the same plausible clock

entry. Moreover, it must be noted that in general the size of the plausible clock is independent of the number of client and of object entities in the system.

Rules to manage t_i/tx_i :

- R1 Each time a process sends a message, it timestamps the message m with the current value of its plausible clock, denoted $m.t$.
- R2 Each time a client c_i writes, it increments its plausible clock entry $t_i[i \text{ modulo } n]$, i.e. $t_i[i \text{ modulo } n] := t_i[i \text{ modulo } n] + 1$.
- R3 Each time a client c_i receives a response message m to a read request, it updates its plausible clock with the timestamp piggybacked by m , i.e. $\forall k t_i[k] := \max(t_i[k], m.t[k])$.
- R4 Each time an object manager receives a write request message m from c_j , if $tx_i[j \text{ modulo } n] < m.t[j \text{ modulo } n]$ then it updates its plausible clock tx_i , i.e. $\forall k tx_i[k] := \max(tx_i[k], m.t[k])$.

4.2. Protocol Behavior

When a client c_i wants to execute a write operation $w_i(x)v$, it increments its entry of the plausible clock t_i and sends an update message corresponding to $w_i(x)v$ to all object entities. A message $m_{write}(v, t)$ corresponding to a write operation, later sometimes referred as *write message*, contains the value v to be written and the value t of the plausible clock at c_i at the time the message was sent.

```

WRITE(v)
1   $t_i[i \text{ modulo } n] := t_i[i \text{ modulo } n] + 1;$ 
2  repeat
3    for ( $1 \leq j \leq n$ ) send [ $m_{write}(v, t)$ ] to  $x_j$ 
4    until [ $\text{receipt}(ack_{m_{write}(v, t)})$  from  $f$   $x_j$ ];
5   $cache := v$ 

```

Figure 2. Write procedure performed by client process c_i

Moreover, because of fair-loss links, client process c_i sends $m_{write}(v, t)$ to all object entities until an ack is received from f object entities, lines 2, 3 and 4 of write procedure in Figure 2. In this way, when c_i completes its write operation $w_i(x)v$, at least f object entities have received $m_{write}(v, t)$. The value written is then stored in c_i 's cache, line 5 of write procedure in Figure 2.

When a client c_i wants to read, it repeatedly sends its read request to all object entities until responses are collected from f different object entities, lines 2-13 of read procedure in Figure 3. A message $m_{read}(num_{seq}, t, cache)$, corresponding to a read, later sometimes referred as *read*

message, contains the sequence number of the request, num_{seq} , the current values of the plausible clock at c_i , namely t , and the current value of c_i 's cache. Due to network delays and retransmission, num_{seq} is necessary to allow a client to discard old responses when received.

In detail, due to fair-loss links client c_i repeatedly sends a read request to all object entities until a response is received, lines 3, 4, 5 of read procedure in Figure 3. When a response is received from x_h , c_i checks if it already received a response corresponding to the current request from x_h , line 6 of read procedure in Figure 3. If no responses for the current request were previously received from x_h , the message is processed by c_i : it tracks an ack more, line 7 of read procedure in Figure 3; it checks if the value stored is a new one w.r.t. to the one in c_i 's cache and if so c_i 's cache and control structures are updated, lines 9-10 in Figure 3. c_i stops to send such a request when one of the following conditions holds: it has received a response from f distinct object entities or it has received a value different from the one stored in its cache.

```

READ ( $x$ )
1   $num_{seq} := num_{seq} + 1;$ 
2  while ( $ack < f$ )
3    repeat
4      for ( $1 \leq j \leq n$ ) send [ $m_{read}(num_{seq}, t, cache)$ ] to  $x_j$ 
5    until [ $receipt(m_{res}(num_{seq}, tx_h, v))$  from  $x_h$  with  $h \in [1..n]$ ];
6    if ( $ack[h] = false$ ) then
7       $ack[h] := true;$ 
8       $ack := ack + 1;$ 
9      if ( $tx_h \neq t$ ) then
10        $cache := v;$ 
11        $\forall k \ t_i[k] := max(t_i[k], tx_h[k]);$ 
12        $ack := f;$ 
13     end if
14   end if
15 end while
16  $ack := 0;$ 
17 for ( $1 \leq j \leq n$ )  $ack[j] := false;$ 
18 return( $cache$ )

```

Figure 3. Read procedure performed by client process c_i

Then, client c_i waits for f response messages by the object entities or to receive a message containing a value different from the one in c_i 's cache. In this last case, it updates its plausible clock t_i in the following way: $\forall k \ t_i[k] := max(t_i[k], m.t[k])$. It stores the new value read in its cache. Finally, the value is return.

When an object manager x_i receives a request of write by a client c_j , it verifies if the write operation has to be considered obsolete w.r.t. \mapsto_{co} , line 2 of write thread in Figure 4. Then, if the write is considered obsolete, x_i discards the message otherwise it applies the value to its local memory and it synchronizes its plausible clock with the one piggy-backed by the write message, lines 3,4 and 5 of write thread in Figure 4. The variable $last$ stores the identifier of the plausible clock entry that was last updated. In any case, it

sends back an ack to client process c_j , line 7 of write thread in Figure 4.

```

1  when ( $receipt(m_{write}(v, t))$  from  $c_j$ ) do
2    if ( $(t[j \text{ modulo } n] > tx_i[j \text{ modulo } n])$ )
3      then  $x := v;$ 
4       $\forall k \ tx_i[k] := max(tx_i[k], t[k]);$ 
5       $last := j \text{ modulo } n;$ 
6    end if
7    send [ $ack_{m_{write}(v, t)}$ ] to  $c_j$ 

```

Figure 4. x_i 's write thread

When an object manager x_i receives a request for a read operation by client process c_j , it has to check causal consistency. If the value of the object manager is causally precedent the one of the client, the object manager simply sends back a response with the values previously sent by the c_j in the current read request, line 4 of read thread in Figure 5.¹ Otherwise, it replies with its value of x and the value of its plausible clock, line 3 of read thread in Figure 5. In any case, it finally answers to the request with a value that may be the one sent by c_j itself in the read request, or a more recent one w.r.t. \mapsto_{co} .

```

1  when ( $receipt(m_{read}(num_{seq}, t, val))$  from  $c_j$ ) do
2    if ( $tx_i[last] > t[last]$ )
3      then send [ $m_{res}(num_{seq}, tx_i, v)$ ] to  $c_j$ 
4      else send [ $m_{res}(num_{seq}, t, val)$ ] to  $c_j$ 

```

Figure 5. x_i 's read thread

A response message $m_{res}(num_{seq}, tx_i, v)$ for a read contains: i) the sequence number of the read request ii) the value of x_i 's plausible clock at response time, tx_i , and iii) the value v to be returned by the read operation.

Figure 6 depicts a simple scenario, where client process c_2 writes the value a subsequently read by another client c_1 .

In Figure 6, object entities values are depicted every time they change and since in this scenario we do not consider memory losses, a value stored is not lost. Notice that, the scenario in Figure 6 also point out that some messages may be lost. This is due to the fact that we consider fair-loss links.

4.3. Correctness Proofs

In this section we first prove that t is a plausible clock capturing \mapsto_{co} and then we prove the correctness of the algorithm we present in section 4.2 to implement a weakly-persistent causal object.

¹It must be noted that in such a case, a refresh purpose might be considered for a read operation, that is the object manager could update its local structure, logical clock and cache, treating the read as a write. This may improve the availability of values written.

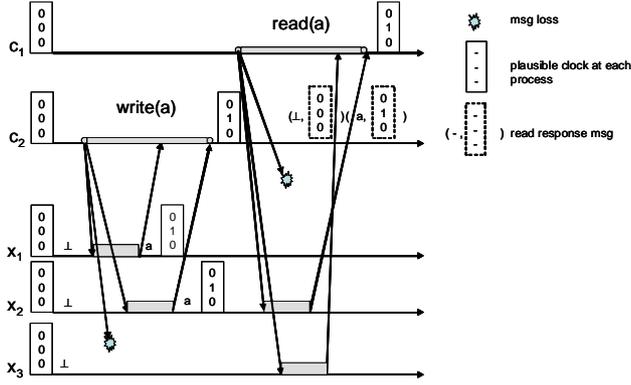


Figure 6. A scenario generated by the implementation of the causal object described in Section 4.

t is a plausible clock capturing \mapsto_{co} Given a write operation $w_i(x)v$, according to line 2 of write procedure in Figure 2, such a write operation is associated with a logical clock t , denoted $t.w_i(x)v$. We have to prove that given two write operations $w_i(x)v$ and $w_j(x)v'$ such that $w_i(x)v \mapsto_{co} w_j(x)v'$, then $t.w_i(x)v < t.w_j(x)v'$. On the other hand, according to the properties of plausible clocks, $t.w_i(x)v < t.w_j(x)v'$ means that one of the following case arises: 1) $w_i(x)v \parallel w_j(x)v'$ or 2) $w_i(x)v \mapsto_{co} w_j(x)v'$.

Notation $w \mapsto_{co}^k w'$ with $k \geq 1$ means that there exists a sequence of k \mapsto_{co} relations $w \mapsto_{co} w_1 \mapsto_{co} \dots \mapsto_{co} w_h \mapsto_{co} w_{h+1} \mapsto_{co} \dots \mapsto_{co} w_{k-1} \mapsto_{co} w_k \mapsto_{co} w'$ and for any relation $w_h \mapsto_{co} w_{h+1}$ does not exist a write operation w'' such that $w_h \mapsto_{co} w'' \mapsto_{co} w_{h+1}$.

Observation 1. At each client c_i , t_i does not decrease.

Observation 2. w is the k^{th} write operation invoked by the client process $c_i \Rightarrow t[i \text{ modulo } n].w(x)v \geq k$.

Proof. Let $w(x)v$ be the k^{th} write operation invoked by the client process c_i . Two possible cases arise:

1) at the time of $w(x)v$ invocation, c_i has not yet executed a read operation. Thus $t[i \text{ modulo } n].w(x)v$ is equal to k according to line 1 of write procedure in Figure 2 and the fact that the initial value of the plausible clock at c_i is $[0, \dots, 0]$.

2) at the time of $w(x)v$ execution, c_i has executed at least one read operation. There are two possible cases: i) c_i reads a value written by itself, thus it does not update its plausible clock and we are again in case 1); ii) c_i reads a value written by another client process. According to line 11 of read procedure in Figure 3, c_i synchronized its clock t with tx , the one sent by the object manager in its response to such a read request, lines 2,3 of read thread in Figure 5. Moreover,

for line 11 of the read procedure in Figure 3, the resulting value of t is not minor than the value of t before such a synchronization. Thus, since $w(x)v$ is the k^{th} write executed by c_i and due to line 1 of write procedure in Figure 2 and to the fact that when a client reads, its plausible clock does not decrease, we have that $t[i \text{ modulo } n].w(x)v \geq k$. \square

Now to prove that t is a plausible clock capturing \mapsto_{co} , we have to prove that: $\forall w_i(x)v, w_j(x)v' : w \neq w', w_i(x)v \mapsto_{co} w_j(x)v' \Rightarrow t.w_i(x)v < t.w_j(x)v'$.

Lemma 1. $\forall w_i, w_j \in H : w_i \neq w_j, (w_i \mapsto_{co} w_j \Rightarrow t.w_i < t.w_j)$

Proof. Let us consider the notation $w_i \mapsto_{co}^k w_j$. The proof is by induction on the value of k .

Basic step. Given two write operations w_i and w_j such that $w_i \mapsto_{co}^0 w_j \Rightarrow t.w_i < t.w_j$. This means that $w_i \mapsto_{co} w_j$ and \nexists a write w' such that $w_i \mapsto_{co} w'$ and $w' \mapsto_{co} w_j$.

We distinguish two cases:

(1) $i = j$. This means that w_i and w_j have been executed by the same client process c_i . Each time a client process executes a write operation, it performs the write procedure in Figure 2. According to line 1 of Figure 2, each time c_i executes a write operation, it increments its corresponding entry of t . Due to Observation 1, if w_i precedes w_j in c_i program order then $t.w_i < t.w_j$. Therefore the claim follows.

(2) $i \neq j$. There exists a read operation invoked by the client process c_j , denoted $r_j(x)$, such that $w_i(x)v \mapsto_{ro} r_j(x)v$ and $r_j(x)v \mapsto_{po} w_j(x)v'$. In detail, c_j can read the value written by c_i because i) c_i has invoked $w_i(x)v$ and at least a majority of object managers have applied $w_i(x)v$ and ii) one of such object managers has answered to c_j read request. Without loss of generality, let us assume that x_k is the object manager that implements points i) and ii). Then, according to line 4 of the write thread in Figure 4, after having applied $w_i(x)v$, tx_k is \geq than $t.w_i(x)v$. Subsequently, c_j reads the value written by $w_i(x)v$. This means that:

- when x_k has received the read message m of c_j , its local value of x was v , that is the value written by $w_i(x)v$. Then according to line 4 and 5 of write thread in Figure 4 and to lines 2, 3 of read thread in Figure 5, x_k sends to c_j a response message $m_{res}(num_{seq}, v, tx_k)$ with $tx_k \geq t.w_i(x)v$.
- when c_j delivers $m_{res}(v, tx_k, num_{seq})$ according to lines 10, 11, 12, 2 and 18 of read procedure in Figure 3, c_j updates its t_j and its cache with the corresponding values piggybacked by $m_{res}(num_{seq}, v, tx_k)$ and then it returns the value to be read, that is v .

Then after the read operation $t_j \geq tx_k$ that is $t_j \geq t.w_i(x)v$. Moreover, it must be noted that i) for observation 1, t_j never decreases and ii) when c_j writes $w_j(x)v'$,

t_j is incremented, (line 1 of write procedure in Figure 2). Then since $w_j(x)v'$ is executed by c_j after the execution of $r_j(x)v$ the claim follows, that is $t.w_i(x)v < t.w_j(x)v'$.

Inductive Step. $w_i \mapsto_{co}^{k>0} w_j$ then: (i) $\exists w' : w_i \mapsto_{co}^{k-1} w'$. By the inductive hypothesis we have: $t.w_i < t.w'$, and (ii) $w' \mapsto_{co}^1 w_j$. Because of *Basic Step* $t.w' < t.w_j$. From (i) and (ii), it follows: $t.w_i < t.w_j$. \square

Object Correctness Proofs

Property 1 (Causal Ordering). *Given two write operations $w(x)v$ and $w(x)v'$ if $w(x)v \mapsto_{co} w(x)v'$, then a client process c_i that reads both values, executes $r_i(x)v$ and then $r_i(x)v'$.*

Proof. Roughly speaking, we have to prove that given two write operations $w(x)v$ and $w(x)v'$ if $w(x)v \mapsto_{co} w(x)v'$, then a client process c_i that reads both values, reads v and then v' . Thus, let us assume that a client c_i has executed $r_i(x)v'$. This means that for lines 4-6 of write thread in Figure 4 and line 11 of read procedure in Figure 3, the logical clock of c_i after the execution of the read is $t_i \geq t.w(x)v'$. Then, when subsequently c_i invokes another read operation, for what said and for observation 1, c_i inserts in the corresponding request message a timestamp $t_i \geq t.w(x)v'$. By contradiction, assume that there is an object manager x_k that responds to that request with $m_{res}(v, tx_k)$, then according to lines 2, 4 and 5 of write thread in Figure 4 and line 3 of read thread in Figure 5, $tx_k[last] = t.w(x)v[last]$. But for lemma 1 $w(x)v \mapsto_{co} w(x)v'$ implies $t.w(x)v < t.w(x)v'$. This means that $\forall k t.w(x)v[k] \leq t.w(x)v'[k]$. This contradicts line 2 of read thread in Figure 5. Thus when x_k receives the read request of c_i with timestamp $t.w(x)v'$ it sends back the value of c_i 's previous request, that is v' line 4 of read thread in Figure 5. \square

Property 2 (Weak Persistency). *If a value v is written infinitely many times, then a client process that read infinitely many times, eventually read v , a concurrently written value or a subsequent one w.r.t. \mapsto_{co} .*

Proof. If a value v is written infinitely many times, due to assumption 1, lines 2-4 of write procedure in Figure 2, line 1 of write thread in Figure 4 and the properties of the plausible clock systems (lemma 1), there is a time after which $f' = \lceil 2n + 1/3 \rceil$ object entities stop to lose their memory and thus v or a value that is causally concurrent with v or a more recent one is permanently stored. This means that a client process that read infinitely many times, will read one of such values due to lines 2-12 of read procedure in Figure 3 and to line 2 and 4 of read thread in Figure 5.

According to this, we have to prove that given n object entities, if a value is stored by $f = \lceil 2n/3 \rceil$ object entities, provided that $f' = \lceil 2n + 1/3 \rceil$ object entities do not suffer memory losses, the value written may be retrieved if not

overwritten. For sake of simplicity, let us consider the case in which there are no concurrent or more recent value written w.r.t. \mapsto_{co} than the value v .

When the write $w(x)v$ terminates, at least f object entities have stored the value, for line 4 of write procedure in Figure 2, line 2 of write thread in Figure 4 and for the properties of the plausible clocks and the assumption of no causally concurrent or more recent write operation. Among these, at most $n-f'$ may lose its status and thus value v , returning to the initial value \perp . Notice that trivially $n - f' < f$.

Let us now consider the worst case, that is the value is stored by the $n-f'$ object entities that subsequently lose their status. Thus summarizing, $n-f'$ object entities do not store the value v , $n-f'$ object entities store and subsequently lose value v and the remaining object entities permanently store such a value.

When subsequently a client process c_i invokes a read request, it waits for a response from f object entities. In the worst case, c_i receives a response by the $2n-f-f'$ object entities that do not have value v . But since it waits for f responses, we are sure that there is at least one response piggybacking value v . In fact, let us notice that $2n - f - f' = 2n - \lceil 2n/3 \rceil - \lceil 2n + 1/3 \rceil \leq 2n - 2n/3 - 2n + 1/3 = 2n - 1/3 < 2n/3 \leq \lceil 2n/3 \rceil$. Since $f = \lceil 2n/3 \rceil$, the proof follows from the fact that $2n - f - f' < f$. In other words, in order to reach f responses, c_i needs a response sent by an object entity that does not belong to the $2n - f - f'$ without the value v . \square

Property 3 (Termination). *Each operation invoked by a correct client eventually completes.*

Proof. • **Write.** Let c_i be a correct client that issues a write operation $w_i(x)v$. Then according to line 3 of write procedure in Figure 2, $w_i(x)v$ completes when c_i receives an ack from f object entities, otherwise it loops into lines 2 and 3 of write procedure in Figure 2. Then we have to prove that if a correct client c_i invokes a write $w_i(x)v$ eventually f $ack_{m_{write}(v,t)}$ are received by c_i . This is ensured by assumption 1 and line 6 of write thread in Figure 4.

• **Read.** Let us now consider the case of a read operation. A read operation completes if f response messages are received, lines 2, 4, and 8 of read procedure in Figure 3. Then we have to prove that if a correct client c_i invokes a read $r_i(x)v$ eventually an ack from f x_k is received by c_i . This is ensured by assumption 1 and lines 3, 4 of read thread in Figure 4. \square

Property 4 (Validity). *If a read operation invoked by a client process c_i returns a value v , then there exists a client process c_j that invoked the write of v .*

Proof. The proof follows by lines 1, 3 of write thread in Figure 4, lines 3-5, 10 and 18 of read procedure in Figure

3, to the read thread in Figure 5 and to the property of *no creation* of fair loss channels, [21]. \square

5. The case of sequential consistency

In this section we point out how we can adapt algorithm presented in section 4 to implement a sequential consistent shared object. A read/write shared object is sequential consistent if for any generated history H , it is possible to find a sequence S containing all the operations in H such that 1) each read operation returns the last value written according to S and 2) for every client process c_i , for every pair of operations o_1 and o_2 executed by c_i such that o_1 precedes o_2 in c_i program order than o_1 precedes o_2 in S .

In detail, instead of using a plausible clock, we use as timestamps, pairs composed by a scalar clock, i.e. a Lamport clock, and process identity. We assume a total order on client process identities. According to this, given two timestamps $t_1 = (l_1, id_1)$ and $t_2 = (l_2, id_2)$, we have that $t_1 < t_2$ if $l_1 < l_2$ or $l_1 = l_2$ and $c_1 < c_2$. In order to guarantee sequential consistency, we use a deterministic rule to totally ordering concurrent write operations, e.g. operations with the same scalar clock are ordered according to the process identifier. As an example, let us consider the following two write operations $w_1(x)a$ and $w_2(x)b$ whose timestamps are respectively $(1, 1)$ and $(1, 2)$. We have that each object entity applies before $w_1(x)a$ and then $w_2(x)b$. This means that an object manager that previously received $w_2(x)b$, will discard $w_1(x)a$ when received. We analogously impose the ordering when a client process reads such values.

6. Related Works

Read/write objects are building blocks to implement several distributed services, i.e. distributed shared memory, distributed directory lookup services, shared boards and so on. Many consistency criteria have been proposed in order to the define objects semantics, e.g. from more to less constraining ones: Atomic [17], Sequential [16], Causal [3] and PRAM [18]. Read/write atomic objects (registers) have been the most studied since they offer to processes the illusion of accessing the object once at time. On the other hand, atomic consistency requests object state persistency thus making atomic object implementations more expensive w.r.t. weaker consistency criteria. Attiya et al. in [6] give the definition of persistency for a single writer/multi-readers atomic register, that is: once a process reads a particular value, then, unless the value of this register is changed by a write, every future read of this register may retrieve such a value, regardless of process slow-down or failure. Herlihy et al. in [14] formalize the concept of persistency for a multi-writer/multi-reader atomic object. In

a distributed message-passing system where processes may fail by crashing, implementations of atomic objects have to cope with the difficulty of providing object state continuity when processes fail. Attiya et al. in [6] propose an implementation for single-writer/multiple-reader register provided that a majority of processes do not crash. Lynch et al. in [20], extend this last work to multiple-writer/multiple-reader registers adopting a more general quorum-based approach. Their solution also tolerates quorums on-line reconfigurations. Some quorum-based solutions have been also proposed to implement atomic objects in dynamic systems where participants may join, leave and crash during the computation, [12, 22], [10]. Instead of using quorums, in [9] Friedman implements an atomic object on top of a virtually synchronous communication layer. In [9], Friedman also investigates sequential and causal consistent shared objects.

On the other hand, in dynamic systems where processes may join and leave at any time and arbitrarily fast, objects implementations are not persistent by nature. To circumvent this problem, Lynch et al. [19] propose a solution to implement atomic consistency when the system is quiescent. Friedman et al. in the context of peer-to-peer systems propose what they call a *semi-reliable unified storage* abstraction [11]. It is interestingly to notice that they implement a notion of atomic consistency restricted to uninterrupted partial execution. An uninterrupted partial execution is a collection of sequences of read and write operations, each one by a different process, such that during their execution there are no failure and the set of processes do not change. On the other hand, in order to guarantee consistency all the time regardless the dynamism of processes, we implement a shared object with a weaker semantics, that is causal consistency [3]. Moreover, we guarantee persistency of value written only during quiescent periods, through the *weak persistency*.

To cope with the complexity of dynamic systems, we exploit the idea proposed by Chen et al. in [8] to solve fault-tolerant mutual exclusion problem in dynamic systems. In detail, the object is implemented by a fixed set of virtual servers that may suffer memory losses. A memory loss abstracts the fact that a virtual server is incarnated by a process that may crash and be replaced by a new process that is not able to retrieve any state the crashed process pass through. It is like considering a fixed set of servers that may crash and recover but such that after recovering completely lose their previous state. Guerraoui et al. in [13], point out that atomic registers may be implemented in a crash-recover model provided that i) a majority of processes never crash or eventually recover and never crash again and that ii) given a write operation $w(x)v$, at least a majority of processes log (i.e. store to stable storage) the value v before the write operation returns. Thus, they extend the atomicity consistency criteria

defined for multi-writer/multi-reader register in a crash-stop model by providing two new criteria: *persistent atomicity*, to capture the fact that traditional atomicity has to persist through the crashes and *transient atomicity* that does not guarantee atomicity in between crashes.

Finally, in order to track causality order relations between operations, we implement a plausible clock system that is an adaptation of *R-Entries vector clock system (REV)* proposed by Ahamad et al. in [5]. Plausible clocks were also used by Ram et al. in [15] to implement a causal memory in a mobile environment. Their system model, however, differs from ours since they consider a fixed set of correct physical master sites and a set of mobile hosts.

7. Conclusions

In this paper, we focused on the problem of implementing shared objects over a highly dynamic asynchronous message passing system characterized by infinitely many processes. We implemented the object by a fixed set of virtual servers, each one incarnated at each time by a single process. A virtual server may suffer *memory loss*: when the process currently incarnating a virtual server crashes (or leaves), a new process replaces the old one but it is not able to retrieve the states the crashed process passed through. To capture a possible infinite sequence of processes incarnating a virtual server we have assumed a *number of memory losses arbitrarily large*. Since this failure model is too weak for implementing traditional atomic objects, we consider a weaker consistency criterion, namely causal consistency [3]. Differently from atomicity, causal consistency may tolerate an infinite number of memory losses, however it may be useful provided that some form of values persistency is ensured. According to this, we defined a persistency property, namely *weak persistency*, and we proposed an algorithm, implementing a so called weak-persistent causal object. This object has the desirable property of not violating causal consistency during the periods in which processes that implement it continuously join and leave. Moreover, the implementation does its best to provide the latest causal consistent state to clients.

Finally, during *quiescent periods* (process joins and leaves subside) persistency and causal consistency is provided to clients.

Acknowledgements

We like to thank Jean-Michel H elary and Michel Raynal for suggestions on this work.

References

- [1] M.K. Aguilera, A Pleasant Stroll Through the Land of Infinitely Many Creatures. *ACM SIGACT News, Distributed Computing Column*, 35(2):36-59, 2004.
- [2] M.Ahamad, R. John, P. Kohli and G. Neiger. Causal Memory Meets the Consistency and Performance Needs of Distributed Application!. *EW 6: Proceedings of the 6th workshop on ACM SIGOPS European workshop*, 45-50, 1994.
- [3] M. Ahamad, G. Neiger, J.E. Burns, P. Kohli and P.W. Hutto. Causal Memory: Definitions, Implementation and Programming. *Distributed Computing* 9(1): 37-49, 1995.
- [4] M. Ahamad, M. Raynal and G. Thia-Kime. An adaptive architecture for causally consistent distributed services. *Distributed System Engineering*, 6: 63-70, 1999.
- [5] M. Ahamad, F. J. Torres-Rojas. Plausible clocks: constant size logical clocks for distributed systems. *Distributed Computing* 12: 179-195, 1999.
- [6] H. Attiya, A. Bar-Noy and D. Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM*, 42(1):124-142, January 1995.
- [7] H. Attiya and J. Welch. *Distributed Computing* (second edition), Wiley, 2004.
- [8] W. Chen, S. Lin, Q. Lian, and Z. Zhang. Sigma: A fault-tolerant mutual exclusion algorithm in dynamic distributed systems subject to process crashes and memory losses. In *Proceedings of the 11th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'2005)*, Changsha, Hunan, China, December 2005.
- [9] R. Friedman. Using Virtual Synchrony to Develop Efficient Fault Tolerant Distributed Shared Memories. Technical Report 95-1506, Department of Computer Science Cornell University, Ithaca, NY, 1995.
- [10] R. Friedman, M. Raynal, C. Travers. Two Abstractions for Implementing Atomic Objects in Dynamic Systems. In *proceedings of the 9th International Conference on Principles of Distributed Systems (OPODIS 2005)*, Pisa, Italy, December 2005.
- [11] R. Friedman, M. Raynal. Modularity: A First Class Concept to Address Distributed Systems. Technical Report **PI-1707**, IRISA, Rennes, 2005.
- [12] S. Gilbert, N. Lynch, and A. Shvartsman. Rambo II: Rapidly reconfigurable atomic memory for dynamic networks. In *Proc. 17th Intl. Symp. on Distributed Computing (DISC)*, pages 259-268, June 2003.
- [13] R. Guerraoui and R. Levy. Robust emulations of shared memory in a crash-recovery model, technical report. <http://lpdwww.epfl.ch/publications>, 2004.
- [14] M. Herlihy and J. Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. on Programming Languages and Systems*, 12(3):463-492, 1990.
- [15] D. Janaki Ram, M. Uma Mahesh, N. S. K. Chandra Sekhar, Chitra Babu: Causal Consistency in Mobile Environment. *Operating Systems Review* 35(1): 34-40 (2001)
- [16] L. Lamport. How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs. *IEEE Transactions on Computers* **28(9)**, 690-691(1979).

- [17] L. Lamport. On Interprocess communication; part I: Basic formalism. *Distributed Computing*, 1(2):77-85, 1986.
- [18] R. Lipton, J. Sandberg. PRAM: a Scalable Shared Memory. Technical Report **CS-TR-180-88**, Princeton University (1988).
- [19] N. Lynch, D. Malkhi, and D. Ratajczak. Atomic data access in content addressable networks. In proceedings of the First International Workshop on Peer-to-Peer Systems (2002).
- [20] N. Lynch and A. Shvartsman. Robust Emulation of Shared Memory Using Dynamic Quorum-Acknowledged Broadcasts. Symposium on Fault-Tolerant Computing (1997).
- [21] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publisher, San Mateo, CA, 1996.
- [22] N. Lynch and A. Shvartsman. RAMBO: A reconfigurable atomic memory service for dynamic networks. In Proc. 16th Intl. Symp. on Distributed Computing (DISC), pages 173–190, Oct. 2002.

Unconscious Eventual Consistency with Gossips

Roberto Baldoni¹, Rachid Guerraoui^{2,3}, Ron R. Levy²,
Vivien Quéma¹ and Sara Tucci Piergiovanni¹

¹DIS, Università di Roma “La Sapienza”, 00198 Roma, Italy

²LPD, EPFL, CH 1015 Lausanne, Switzerland

³CSAIL, MIT, Cambridge, MA 02139, USA

Abstract. We present an update ordering protocol based on gossiping for geographically distributed replicas. We target large scale distributed systems without central authority. Updates are disseminated epidemically and continuously applied to the replicas despite network partitions and asynchrony. No two updates are ever performed in different orders, but gaps might occur during periods of unreliable communication. These gaps are filled whenever connectivity is provided: consistency is then eventually ensured, but without any conscious commitment. This unconsciousness is the key to tolerating perpetual asynchrony. A simulation study shows that the protocol is scalable and achieves high throughput under load.

1 Introduction

Background. A new class of so-called *interactive* distributed applications is emerging : these include distributed virtual environments, multi-player games, interactively steered scientific applications, collaborative design systems, etc [3]. These applications may need to run in a wide area asynchronous environment with widely distributed users and resources and no central authority. On-line games might for instance involve thousands of users [20]. In such settings it is important for each user to have access to a local copy (replica) of every object of interest. This is key to allowing local progress without constantly relying on the network. The main technical challenge is then to maintain some form of consistency among all replicas of the same object [17].

Traditionally, many systems running on local area networks provide so-called single copy semantics (also referred to as atomicity). Single copy semantics gives the user the illusion of accessing a single, highly available instance of an object. Typical solutions require users to access a quorum of replicas, to acquire exclusive locks on data they wish to update or to agree on a total order of updates to be applied at each replica. Total ordering requires each update to be executed at most once at each replica and any two updates to be executed in the same order at all replicas. Unfortunately, maintaining single-copy semantics in a worldwide deployed system is practically very expensive and theoretically impossible [9].

In the kind of wide-area settings we consider, it is necessary to use (*weaker*) ways of ordering updates. This is precisely what eventual consistency [19] provides. It guarantees that whatever the current state of the replica, if no new

updates are issued and replicas can communicate freely for a long enough period, the contents of all replicas eventually become identical. This consistency criterion was claimed to be very efficient in weakly connected environments such as the one targeted by Bayou [19] and OceanStore [15]. From an implementation point of view, the issues to solve in order to guarantee eventual consistency are [17]: (1) *update dissemination*: each update must eventually reach all replicas, and (2) *update ordering*: all updates must be *eventually* applied in the same order at each replica to ensure that the last update is the same at all replicas.

In current solutions [15, 19], update dissemination is based on epidemic protocols. Update ordering [15, 19, 13, 21, 18] is achieved by having replicas deliver updates locally in any order (tentative order) and using *rollbacks* to eventually reach a total order. Total ordering is typically computed *a posteriori* using some form of consensus. This requires a “synchrony island” where agreement can be achieved to ensure that all replicas eventually agree on the exact update order. When that happens, each replica is *conscious* of the fact that total order has been reached.

Contributions. This paper presents a protocol for achieving eventual consistency in large-scale distributed environments. Update dissemination is performed using a classical gossip-based protocol [8]. Our protocol differs from others by the fact that it does not use any form of consensus, even only eventual. Our protocol defines an *a priori* total order that is never explicitly agreed upon among replicas. Updates are disseminated using gossips and subsequently delivered. In the case that some old update arrives after already having delivered subsequent messages, the replica has to roll back to the old state, apply the old update and re-deliver all subsequent messages. This means that, in theory, each replica should keep all delivered updates forever. However, in practice, it is possible to reach consistency with high probability without keeping all delivered updates.

A fundamental aspect of our protocol is that replicas are *unconscious* of when total order is reached, i.e. a replica cannot know of the existence of some old, as of yet undelivered messages. This unconsciousness is the key to reaching eventual consistency even if the network is permanently asynchronous. This characteristic is particularly important for interactive applications based on continuous shared data [3]. It is important for such applications to allow users to access replicas in the face of frequent disconnections.

Our protocol has the following characteristics:

- *Non-blocking* : the protocol enables update delivery even during periods when the network is asynchronous or partitioned.
- *Stability* : the protocol exploits periods of (even partial synchrony) and merging of partitions to reduce the number of rollbacks. (Note that the periods of synchrony are not relied on to reach consistency).
- *Scalability* : the protocol encompasses an autonomic mechanism that guarantees high throughput when the number of broadcasters and/or the rate at which they broadcast updates increase.

Our simulations convey our claim: we show that our protocol achieves reasonable latency during synchronous periods (due to a small number of rollbacks) and achieves high throughput under high load.

Roadmap. This paper is organized as follows. Section 2 presents the ramifications underlying unconscious eventual consistency. Section 3 describes our protocol. A performance evaluation is presented in Section 4. Finally, related work is presented in Section 5, before concluding the paper in Section 6.

2 Ensuring Unconscious Eventual Consistency

Roughly speaking, eventual consistency stipulates that all replicas eventually converge to the same state, i.e. deliver the same set of updates in the same order. It is possible to achieve eventual consistency in an *unconscious* manner by defining an a priori total order on updates. Replicas (called processes in the rest of the paper) deliver updates in their order of arrival, thus not requiring any coordination among processes. Consistency is eventually achieved by using a rollback mechanism to re-order already delivered messages.

Note that eventual consistency requires that all updates eventually reach all processes. Reliable communication is therefore necessary. However, in a large scale environment, ensuring strong reliable communication can be very expensive. Consequently, most solutions [15, 19] rely on epidemic dissemination [11, 4, 7], even if they do not provide strong reliability¹. Therefore, just like [15, 19], our protocol only provides eventual consistency with high probability.

In the rest of the section we discuss the ramifications underlying unconscious eventual consistency with the aim of better understanding our protocol in the next section. In particular, we discuss the ramifications underlying update rollbacks.

2.1 Update Sequencing

Consider a finite and ordered set of processes $\{p_1, \dots, p_n\}$. Each process acts as a *sequencer*; it keeps a local sequence number that is increased before broadcasting a new message (update). Along with the sequence number, each process tags the message m with its id. The resulting message (m, id, seq) is then disseminated to all processes. A total order is defined on these messages using the sequence number and id. More precisely: for any pair of messages m and m' , m precedes m' iff (i) $seq < seq'$ or (ii) $seq = seq'$ and $id < id'$.

Upon reception of a message, a process cannot possibly know if it will ever receive another message preceding it in the total order. It therefore doesn't make sense for a process to wait for other messages. Consequently, processes deliver messages upon reception. If a message m_1 is received after a message m_2

¹ Note that several works such as [12, 14] focus on improving the reliability of epidemic protocols.

preceding it in the total order, a rollback is performed on m_2 . Subsequently, m_1 and m_2 are delivered in the correct (total) order.

Let us illustrate this naive implementation with a simple example. Consider a system with three processes $\{p_1, p_2, p_3\}$. Process p_1 sends two messages $(m_1, p_1, 1)$, $(m_3, p_1, 2)$ and p_3 two messages $(m_2, p_3, 1)$, $(m_4, p_3, 2)$. No other messages are sent. In this case the totally ordered sequence $\{m_1, m_2, m_3, m_4\}$ is not *consecutive* in the sense that no messages with id p_2 are ever sent. We say that the missing messages are *gaps*. These gaps in the sequence lead to some uncertainty. After a process p_i receives all messages $\{m_1, m_2, m_3, m_4\}$, it still does not know if it is missing a delayed message coming from p_2 (to be delivered between m_2 and m_3) or if this message does not exist at all.

2.2 Rollbacks

The drawback of the naive implementation is that there is no mechanism to reduce the number of rollbacks. In particular, with a large number of sequencers, the number of rollbacks in the system drastically increases. We show that the more sequencers are present in the system, the more rollbacks are necessary.

In the naive implementation all processes give sequence numbers to messages. Consider that there are N sequencers in the system identified by $s_1 < \dots < s_N$. Each sequencer sequences k messages. For simplicity of presentation, consider that messages are broadcast using a reliable FIFO broadcast primitive. If $N = 1$, all messages are received in the correct order by all processes. Thus, no rollbacks are necessary. However, with a larger number of sequencers, the number of possible rollbacks increases. Consider the case $N = 2$ with s_1 and s_2 starting to broadcast at the *same time* and *same rate*. Moreover, consider that messages sent by s_2 are systematically received before messages sent by s_1 . Messages arrive at each process in the following order: $(m_2, s_2, 1)$, $(m_1, s_1, 1)$, $(m_4, s_2, 2)$, $(m_3, s_1, 2)$, etc. Consequently, each process needs to rollback k messages (those sent by s_2). Now, if we extend the previous example to a system with $N = m$ sequencers, it is trivial to demonstrate that each process performs $(m - 1) * k$ rollbacks.

We describe in the next section our protocol and we discuss in particular how we exploit merging of partitions and periods of synchrony to reduce the possible number of sequencers and hence reduce the number of rollbacks. In particular, our protocol seeks to ensure that there is only one sequencer in the system and this is ensured if the network is “synchronous enough”.

3 Protocol

This section presents the protocol, starting with a general overview followed by an in depth description.

3.1 Overview

The protocol we propose combines a sequencing service with gossip based message dissemination. Even though it would be easier to implement the sequencing service using a single process, this is impossible for scalability reasons. Instead, our protocol relies on a pool of processes organized in a *coalition*. As depicted in Figure 2, a process wishing to ecBroadcast a message first requests a sequence number from *the coalition it relies on* and then uses gossiping to disseminate the message together with its sequence number.

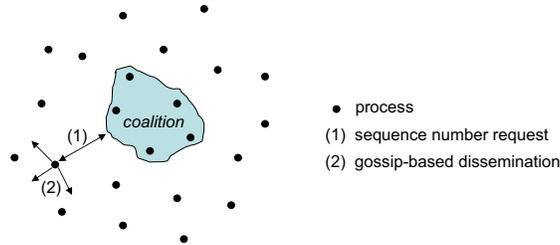


Fig. 1. Basic behavior of the protocol.

Sequencing using coalitions. A coalition c_k is a set of processes (called *members*) acting as a common sequencer. Within a coalition, processes are sorted using their identifiers. We note $c_k[x]$ the x^{th} process in c_k (x is called *rank* of process $c_k[x]$) and we note $\text{card}(c_k)$ the cardinality of coalition c_k . Processes belonging to a coalition issue sequence numbers as follows: let c_k be a coalition and let p_j be a process belonging to c_k , $p_j = c_k[x]$. Process p_j assigns monotonically increasing sequence numbers belonging to the sequence $SN^{c_k[x]} = (sn_n)_{n \in \mathbb{N}}$ with $sn_n = n * \text{card}(c_k) + x$. Along with this sequence number, messages are tagged with the id of the process that issued the sequence number.

Note that the above-described mechanism ensures that a coalition issues distinct sequence numbers. Nevertheless, there is no guarantee that these sequence numbers will be *consecutive*. For instance, consider a coalition made of two processes p_1 and p_2 . Assume that p_1 issues sequence numbers $\{0, 2, 4, 6\}$, and that p_2 issues sequence numbers $\{1, 3\}$. Finally, assume that p_2 crashes; this implies that the coalition will never issue sequence number 5.

Dissemination. We rely on a gossip-based protocol for message dissemination [8]. It has been shown that these protocols are able to ensure high delivery ratios (almost all messages are received by all processes). Moreover, for improving reliability during periods when the network is highly asynchronous or partitioned, the protocol uses a *pull* mechanism that shares similarities with the one proposed in Bayou [19].

Message delivery. Processes try to deliver messages in sequence. This is done by waiting until the preceding messages have been delivered before delivering the current one. However, a process cannot possibly know about all preceding messages for three reasons: (1) there might be other coalitions issuing sequence numbers, (2) the sequence numbers issued by the coalition the process relies on are not necessarily consecutive, and (3) the gossiping mechanism used for dissemination is not reliable. Therefore, a process only waits for a given period of time before delivering received messages. Consequently, a message can be received after consecutive messages have already been delivered. In this case a rollback mechanism is used to undeliver messages and re-deliver them in the correct order. Our experiments show that in the case when only one coalition is present in the system, the number of rollbacks is close to zero.

Coalition creation. If a process p_i that does not rely on a coalition wants to `ecBroadcast` a message, it first tries to discover an already existing coalition. If it does not find one, it creates a new coalition. To do that, process p_i includes itself and some other processes (to get the desired size of the coalition) in a new coalition.

Coalition merging. As explained above, it is desirable to have a single coalition in the entire system. Members of different coalitions get to know each other when they receive messages sequenced by a different coalition. As depicted in Figure 2, if a member p_i of a coalition A receives a message coming from another coalition B , then it builds a new coalition C including all members of A and B . As explained below, the size of the resulting coalition is readjusted after the merger.

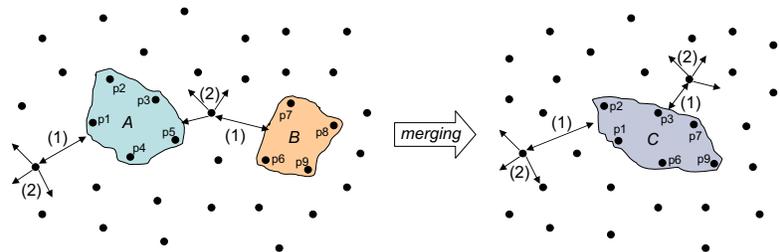


Fig. 2. Coalition merging.

Assuring eventual total order with changing coalitions. Each time a coalition member switches to another coalition (e.g. after a merger) it starts issuing new sequence numbers as explained above. Therefore a process could reissue the same sequence number twice. This problem is solved by adding an

epoch number to each sequenced message. When a process joins a coalition, it associates an epoch number to this new coalition. This epoch number must be greater than the epoch number of the last coalition the process was a member of. For instance, if process p_i was previously a member of c_1 with epoch number e_1 , then a member of c_2 with e_2 and then joins c_3 , the epoch $e_3 = \max(e_1, e_2) + 1$.

Epoch numbers do not change the way processes deliver messages. We just need to change the way the total order on messages is defined such that the epoch number takes precedence over the sequence number and finally the process id.

Self-sizing coalitions. Scalability of the sequencing service is obtained by dynamically adjusting coalition size according to the load on coalition members. The load on a coalition depends on the number of broadcasters in the system and the rate at which they broadcast. These two parameters are often impossible to determine a priori in the target environments. The self-sizing mechanism described in Section 3.3 dynamically modifies the size of coalitions, based on the average number of sequence number requests that coalition members receive during a period of time.

It is important to note that the smaller the coalition, the smaller the probability of gaps appearing in the sequence numbers, and the bigger the coalition, the higher the number of concurrent requests the coalition can handle. When adding a new member to a coalition only the most stable processes are considered. Conversely, the least stable processes are removed first when reducing the size of a coalition. Section 3.3 describes a mechanism that determines the stability of processes according to the number of messages they have delivered.

3.2 Main Protocol

We start by explaining the basic behavior of the protocol. Then, we describe the aging mechanism that allows the selection of stable processes. Finally, we detail the self-sizing mechanism in charge of adjusting coalition size according to system load.

Data structures. Each process p executing the algorithm contains the following set of data structures. *coalition* represents the coalition process p relies on. It is a list of processes. *optimalSize* is the size that the *coalition* must have. *epoch* represents the epoch process p is in. *nextSN* is the next sequence number from the coalition that p relies on and expects to deliver next. *pending* is the list of messages that process p received but did not yet deliver. Each entry in the *pending* list contains $[m, sn, ts]$, where m is the message to be delivered, sn is its sequence number (integrating the process id, epoch number and sequence number attributed by the sequencing service), and ts the time at which message m was received. The *deliveryTimeout* parameter indicates the time process p should wait before delivering the first message in *pending*. All messages that have been delivered so far are stored in the *delivered* list. Finally, *nbOfRetries*

refers to the number of attempts to retrieve a coalition process p must do before creating its own coalition.

Note that for the sake of clarity, some functions (resp. messages) that are described below take a parameter, named *info*, that is a data structure carrying various data on the process that called the function (resp. sent the message). For instance *info.coalition* contains the coalition the process relies on; *info.epoch* carries its epoch; etc.

The isNext() function. To ease the reading of the algorithm, we have isolated the `isNext()` function (Figure 3), whose role is to indicate if a message must be delivered (returns `true`) or if it must stay in the *pending* list. This function enforces the following policy: the protocol can only wait for messages that are sequenced by the coalition the process relies on and at the same epoch as the one the process is currently in. All other messages are delivered as soon as they are received. This choice is motivated by the fact that most of the time (i.e. when the network is “synchronous enough”), there is only one coalition in the system. Note that it would be easy to also wait for messages sequenced by other coalitions simply by having one *nextSN* field per coalition.

```

1: function isNext(sn, ts)
2:   if (sn.pid ∈ coalition) ∧ (sn.epoch = epoch) then
3:     if (sn.number = nextSN) ∨ (ts + deliveryTimeout < getTime()) then
4:       nextSN := sn.number + 1
5:       return true
6:     else
7:       return false
8:   return true

```

Fig. 3. The `isNext()` function.

Algorithm executed by any process. Figure 4 depicts the algorithm executed by any process p_i . The `coalitionUpdate()` aims at updating the knowledge p_i has about existing coalitions. It is called each time a new message is received. It simply changes p_i 's coalition if p_i 's epoch is lower than the epoch of the coalition given in parameter.

Process p_i can use the `ecBroadcast()` function to initiate the broadcast of a message m . This function first gets a sequence number using the `getSN()` function; it then gossips the message together with its sequence number and information about p_i (coalition and epoch); finally, it adds message m to the *pending* list. The `getSN()` function first tries to retrieve a coalition using the

For each process p_i 1: procedure ecBroadcast(m) 2: $\langle sn \rangle :=$ getSN() 3: gossip ($m, sn, info$) 4: $pending.add([m, sn, getTime()])$ 5: function getSN() 6: repeat $nbOfRetries$ times 7: $\langle info \rangle :=$ getCoalition() 8: if $info \neq \emptyset$ then 9: coalitionUpdate($info$) 10: return snRequest() 11: $info.coalition = \{p_i\}$ 12: $info.epoch = epoch + 1$ 13: coalitionUpdate($info$) 14: return snRequest() 15: upon gossip($m, sn, info$) from p_j do 16: coalitionUpdate($info$) 17: $pending.add([m, sn, getTime()])$	For each process p_i 18: upon $pending.first = [m, sn, ts]$ with isNext(sn, ts) do 19: $rollback = \emptyset$ 20: while $m \prec delivered.last$ do 21: rollback($delivered.last$) 22: $rollback.add(delivered.removeLast())$ 23: ecDeliver (m) 24: $delivered.add(m)$ 25: while $rollback \neq \emptyset$ do 26: ecDeliver ($rollback.removeFirst()$) 27: $pending.remove([m, sn, ts])$ 28: procedure coalitionUpdate($info$) 29: if $info.epoch > epoch$ then 30: $coalition := info.coalition$ 31: $epoch := info.epoch$ 32: $nextSN := 0$
---	---

Fig. 4. Algorithm executed by any process p_i .

getCoalition()² function. Then, it uses the snRequest()³ function to get a sequence number from the coalition returned by the getCoalition() function. Note that after $nbOfRetries$ unsuccessful tries, the getSN() function creates a coalition (containing process p_i).

When process p_i receives a gossip message m , it first updates its coalition using the information contained in m ; it then adds m to the *pending* list. Messages stored in the *pending* list are delivered as soon as they are first in the list and that the isNext() function returns true. Note that the delivery of a message may require rolling back and re-delivering previously delivered messages (Lines 19-22 and 25-26).

Algorithm executed by coalition members. The algorithm executed by a coalition member p_i differs from the one described in the previous section by the coalitionUpdate() function. The latter is depicted in figure 5. It encompasses a merging mechanism that aims at reducing the number of concurrent coalitions in order to lead, when the network is synchronous enough, to a system with only one coalition. Its behavior is the following: when the coalition given in parameter is the same as p_i 's coalition, the function simply updates p_i 's epoch if it is lower than the one passed as a parameter. When coalitions differ, the function merges the two coalitions and uses the size() function to try to reach the coalition's

² For space reasons, the getCoalition() function is not described. This function either returns the coalition p_i relies on (if such a coalition exists), or broadcasts a "coalition request" message to discover a coalition.

³ For space reasons, the snRequest() function is not described. This function simply requests a sequence number from one member of the coalition p_i relies on. Note that, each time this function is invoked, it sends the request to a different member in order to balance the load over all coalition members.

optimal size. This function either truncates the coalition using the `truncate()` function, or adds processes returned by the `getProcess()` function. Several implementations of the `truncate()` and `getProcess()` functions can be done. A basic implementation of the `truncate()` function is to remove processes with highest IDs. The `getProcess()` function can use a basic implementation that randomly return processes chosen among the neighbors of p_i . In section 3.3, we present other implementations of these functions that aim at selecting stable processes.

For each coalition member p_i	For each coalition member p_i
<pre> 1: procedure coalitionUpdate(info) 2: if info.coalition = coalition then 3: if info.epoch > epoch then 4: epoch := info.epoch 5: nextSN := 0 6: else 7: merge(coalition, info.coalition) 8: size(coalition) 9: epoch := max(epoch, info.epoch) + 1 10: nextSN := 0 </pre>	<pre> 11: procedure merge(c1, c2) 12: c1 := c1 ∪ c2 13: procedure size(c) 14: if card(c) > optimalSize then 15: truncate(c) 16: else 17: while (card(c) < optimalSize) ∧ 18: hasMoreProcesses() 19: c := c ∪ getProcess() </pre>

Fig. 5. Algorithm executed by any coalition member p_i .

3.3 Protocol Extensions

This section presents three extensions to the protocol. These extensions aim at handling faults affecting coalition members, reaching stability, and ensuring scalability.

Handling faults in coalitions. The protocol presented in the previous section does not handle faulty coalition members. This does not affect the correctness of the protocol, but it alters its *latency*. Indeed, faulty members are not removed from coalitions⁴. The consequence is that some sequence numbers will never be emitted (recall that each member assigns distinct sequence numbers based on its rank in the coalition), thus leading to a frequent reliance on the *deliveryTimeout* to deliver messages.

To solve this problem, we propose to run a *heartbeat* protocol among coalition members (Figure 6). Each member periodically (δ) sends a *PING* message to other members in the coalition. Members maintain two data structures: *alive* is the list of processes from which a *PING* message has been received. This list is reset periodically. *suspected* is the list of processes that the member suspects. This list is built by adding members of the coalition that are not in *alive* after ($2 * \delta$) *ms* (Line 7), and by adding members suspected by other members (Line 12).

⁴ Faulty members remain in the coalition until all members fail, which leads to the creation of a new coalition by the `getSN()` function.

Processes that are in the *suspected* list of a process p_i will no longer be added by p_i in a coalition⁵ (Line 17).

The above-described behavior requires some additional comments: the *heart-beat* protocol does not prevent *false suspicions*. On the contrary, once a member is suspected by some process p_i , it will eventually be suspected by all other coalition members (Line 12). Nevertheless, if *suspected* lists were not propagated, coalitions would oscillate as long as one process falsely suspects another member. Moreover, propagating *suspected* lists is not a real issue since (1) timeouts can be set sufficiently large to prevent most cases of false suspicions and (2) it is possible to remove processes from the *suspected* lists after some (long enough) period of time, in order to allow falsely suspected processes to re-integrate coalitions.

For each coalition member p_i	For each coalition member p_i
<pre> 1: <i>suspected</i> := \emptyset 2: <i>alive</i> := \emptyset 3: task heartBeat every δ ms 4: send(<i>PING</i>, <i>info</i>) to all $p_j \in$ <i>coalition</i> 5: upon receive(<i>PING</i>, <i>info</i>) from p_j do 6: <i>alive</i> := <i>alive</i> \cup $\{p_j\}$ 7: <i>suspected.add(info.suspected)</i> 8: <i>coalitionUpdate(info)</i> </pre>	<pre> 9: task coalitionMaintenance every $(2 * \delta)$ ms 10: <i>info.epoch</i> = <i>epoch</i> + 1 11: if <i>alive</i> \neq <i>coalition</i> then 12: <i>suspected.add(coalition \ alive)</i> 13: <i>info.coalition</i> = <i>alive</i> 14: <i>coalitionUpdate(info)</i> 15: <i>alive</i> := \emptyset 16: procedure merge(<i>c1</i>, <i>c2</i>) 17: <i>c1</i> := $(c1 \cup c2) \setminus$ <i>suspected</i> </pre>

Fig. 6. Extension for handling coalition members faults.

Reaching stability with aging. This section describes how the protocol's *stability* can be improved by changing the way coalition members are selected. The goal is to select the most *stable* members. A member is said to be stable when it remains in the system for a long period of time⁶. Reaching stability is important because it limits the number of concurrent coalitions that may be created when coalition members fail, thus decreasing the number of required rollbacks.

In the protocol described in the previous section, coalitions are truncated based on members' IDs. The main advantage of this mechanism lies in its determinism: the execution of the `truncate()` function by different coalition members on the same set of processes produces the same subset of processes. Nevertheless, this mechanism has a major drawback: it does not favor *stable* processes. Indeed, all processes have an equal opportunity of becoming coalition members.

⁵ Note that this requires that the `getProcess()` function never returns suspected processes.

⁶ Experimental studies have shown that stable processes are often present in large scale systems such as peer-to-peer systems [10].

This raises the following issue: consider the scenario in which a process p with a very high IDs is “isolated” for some period of time. Process p will create its own coalition (in the `getSN()` function). When other coalitions will discover p ’s coalition, they will merge using the `truncate()` function. Because of its very high ID, p will be selected for staying in the truncated coalitions. As a consequence, these coalitions will change, even if they were not affected by member crashes. This behavior affects the protocol’s *stability*.

To improve the protocol’s *stability*, we propose to use an *aging* mechanism⁷ that shares similarities with the mechanism used to improve the reliability of epidemic broadcast algorithms [8]. The basic idea underlying this mechanism is that each process has an age that reflects the number of messages the process delivered (the age is incremented every N deliveries). Each process stores the age of coalition members and propagates them with each message (in the *coalition* list). Then, the `truncate()` function selects the members with highest age⁸. Eventually, stable processes will have a higher age than all other processes, which guarantees that all coalition members will be stable.

Note that, contrarily to the previously described `truncate()` function, there is no guarantee that two executions of this function by two different coalition members will produce the same result. Indeed, this depends on the knowledge that these two members have about the ages of all coalition members. Nevertheless, this is not an issue because the probability of having different knowledge can be decreased by increasing N . Thus, this doesn’t affect the fact that stable processes will eventually be older than all other processes.

Moreover, note that it is possible to increase the speed at which stability is reached by ensuring that the `getProcess()` function returns “old” processes. Our protocol achieves this by having each coalition member maintain a (short) list of the oldest processes it knows. This list is propagated with each *PING* message.

Improving scalability using coalition self-sizing. In our context, ensuring scalability consists in handling a large number of nodes and guaranteeing high throughput in message deliveries under high load. The protocol described so far already deals with scalability issues by (1) using a gossip protocol to disseminate messages, (2) distributing the sequencer role among several processes (coalition), and (3) balancing the load among coalition members by requesting sequence numbers in a round-robin fashion. Nevertheless, one limitation of the protocol is that it assumes a priori knowledge of the *optimal* coalition size. This optimal size can only be computed if the system size and the broadcast rate are (approximately) known in advance and only slightly vary over time, which is not the general case.

This section describes an extension to the algorithm⁹ that aims at dynamically computing the optimal coalition size. This *self-sizing* mechanism is based on the fact that during a long enough period of time, all coalition members

⁷ For space reasons, we do not provide the pseudo-code of this mechanism.

⁸ In case of equal age, the `truncate()` function selects the member with the highest ID.

⁹ For space reasons, the pseudo-code of this extension is not shown.

experience the same load (due to the round-robin load balancing mechanism). Therefore, computing the optimal size can be done by a specific member (i.e. the member that has rank 0, which we will call the “smallest member”), by simply looking at the load it experienced during the last *sizing period*. If the node is overloaded, it adds processes to the coalition; otherwise, it removes processes. This is the responsibility of the application deployer to decide the maximal load (in terms of request/seconds) a node in the system can support¹⁰.

When two coalitions merge, the optimal size is set to the sum of the optimal sizes of both coalitions. This is the only case when the optimal size can be changed by a member other than the smallest one. In such, it is necessary to determine if the optimal size is the one set by the smallest member or by the process that executed the merger. This decision can easily be done by propagating a *sizing number* together with the optimal size sent in each message. The goal of this sizing number is to easily know if a sizing decision succeeds or precedes another one. This sizing number is incremented by the smallest member each time it changes the optimal size. Moreover, when two coalitions with sizing numbers n_1 and n_2 merge, the sizing number is set to $\max(n_1, n_2) + 1$. Coalition members update their optimal size each time they receive a message with a sizing number greater than the one they currently use.

4 Performance

In this section, we present the performance results obtained by simulating our algorithm. We start by describing the simulation settings and then give the actual performance measurements. The goal of the simulations is to show that the protocol is (1) stable, (2) non-blocking, and (3) scalable.

4.1 Simulation Environment

We simulated our algorithm using the Peersim simulator [1]. Peersim allows cycle-based simulations of distributed algorithms in large-scale environments. Processes in peersim can be connected using arbitrarily complex topologies. In all experiments described in this section, processes are connected using a random graph topology: every process knows a fixed number of random processes. Moreover, processes disseminate messages using an LPBCast-like broadcast protocol [8].

Note that we extended the simulator in order to be able to simulate asynchrony. Incoming and outgoing message queues for inter-process communication were added for this purpose. We can vary the time (i.e. number of cycles) it takes for a message to be transferred from the outgoing queue of the sending process to the incoming queue of the receiving process. In our experiments, this time is

¹⁰ This parameter must fit all nodes in the system. It is thus preferable not to overestimate it. Mechanisms might be added to the algorithm to allow some nodes to handle more requests than others.

bounded by $maxLatency$, and every message transfer takes a random number of cycles ranging from 1 to $maxLatency$.

Finally, we model churn (i.e. continuous joining and leaving of processes) by periodically replacing a percentage of processes. Replacing a node simply consists in resetting all the data structures it contains (except the neighbor table) and generating a new ID for this node. All experiments are run with 1000 processes, with a PING period (δ) of 20 cycles and a sizing period of 40 cycles. All the experiments start with a warm-up phase (first 100 cycles) in which processes progressively join.

4.2 Stability

The first experiment illustrates the fact that the protocol selects stable processes. It consists in simulating 1000 processes that randomly broadcast messages. The self-sizing mechanism was disabled and the optimal coalition size was set to 8. The goal of the experiment is to show how coalitions evolve (i.e. the average number of stable members in each coalition). For the sake of clarity, the average was only computed on coalitions that stayed in the system for longer than 20 cycles.

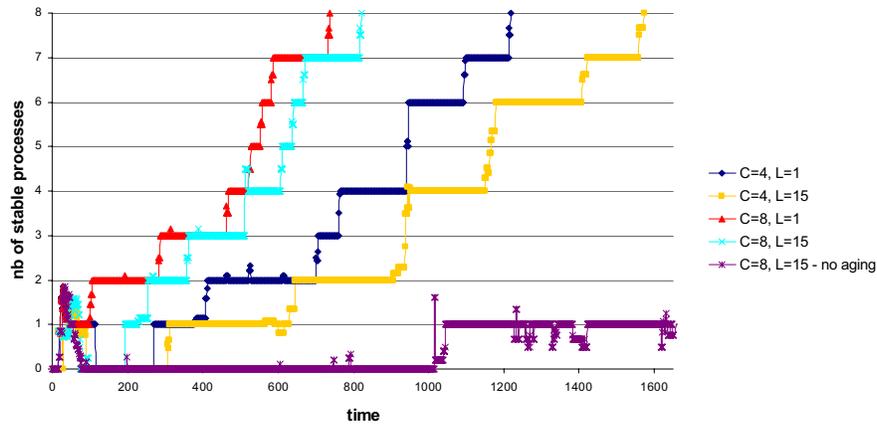


Fig. 7. Stable processes selection.

Figure 7 depicts the average number of stable processes in each coalition as a function of time (i.e. cycle number). We varied both the latency (through the $maxLatency$ parameter) and the churn rate. The $maxLatency$ parameter ranges from 1 to 15; the churn rate ranges from 4% to 8% every 15 cycles. We observe that without any aging mechanism, the protocol does not reach stability (last plot). On the contrary, the aging mechanism ensures that stability is reached (first four plots), i.e. that eventually there will be 8 stable processes in the

coalition. Nevertheless, the speed at which stability is reached depends on the level of asynchrony and churn.

- The stability time increases with asynchrony for two reasons: (1) more time is necessary for coalitions to meet, and (2) asynchrony alters the knowledge that processes have about the age of other processes. Therefore, the protocol has a higher probability of selecting processes that are not stable.
- Increasing churn decreases the time it takes to reach stability. This result might seem surprising, but it can easily be explained by the fact that: (1) unstable members in the coalition have higher probability to fail (and thus to be replaced), and (2) stable processes are proportionally older (and thus have higher probability to be selected).

4.3 Non-Blocking Behavior

The second experiment illustrates the fact that our protocol is non-blocking. In particular, we show that it still provides service during periods where the network is partitioned. The experiment consists in simulating 1000 processes that randomly broadcast messages. The *maxLatency* parameter is set to 10. Moreover, there is no churn. In order to simulate 3 network partitions, we group processes into 3 groups. The interconnection graph is built in such a way that each process has an equal number of (randomly chosen) neighbors in each group. A network partition is simulated by stopping message transfers involving processes that are in different groups.

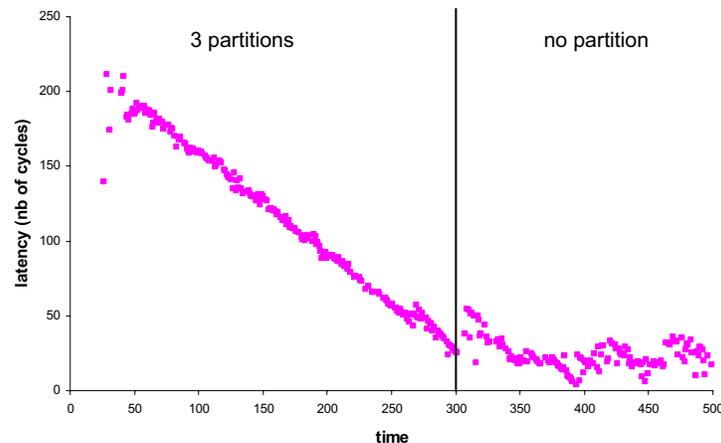


Fig. 8. Average message latency.

Figure 8 plots the average latency of a message broadcast as a function of the time at which the broadcast was initiated. The experiment starts with

three network partitions that merge at cycle 300. As explained in Section 3.1, messages that are not delivered by the gossip primitive are retrieved using a *pull* mechanism. In the depicted experiment, this is the case of most messages sent between cycles 0 and 300. Indeed, our protocol keeps providing service, but the gossip primitive only delivers messages to processes belonging to the same partition as the one the message's broadcaster is in. Other processes wait until the partitions have merged to retrieve these messages using the pull mechanism. This behavior explains the high latency of message broadcasts initiated between cycles 0 and 300. Messages broadcast after cycle 300 have an average latency ranging from 5 to 40 cycles. This is reasonable considering that the maximum latency of a point-to-point communication is equal to 10 cycles.

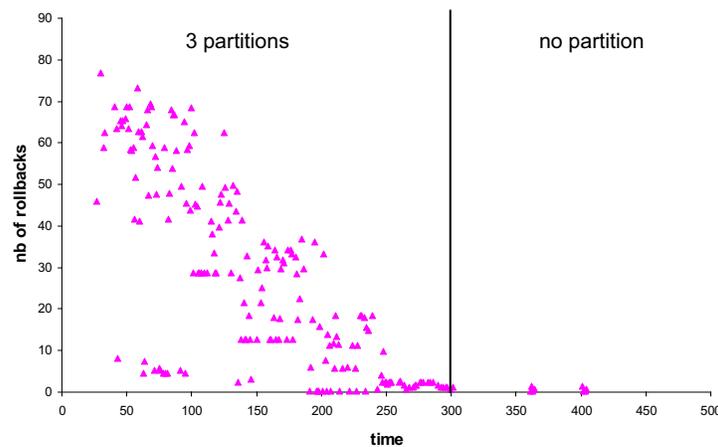


Fig. 9. Average number of rolled-back messages.

Figure 9 plots the average number of rollbacks that were done before delivering a message as a function of the time at which the broadcast was initiated. The experiment is the same as the previously described one. We observe that messages broadcast between cycles 0 and 300 require rollbacks before being delivered. This can be explained by the fact that these messages were previously delivered in the partition of their respective broadcasters. After the network merger, these messages are retrieved using the pull mechanism. Their delivery requires rolling-back part of messages that were delivered during the network partition. We also observe that messages sent after cycle 300 do (almost) not require any rollback before being delivered. This shows that our protocol behaves like a traditional total ordering protocol when the network is synchronous enough.

4.4 Scalability

The last experiment we present demonstrates that the protocol is scalable. In particular, we show that the protocol ensures (almost) constant throughput even during periods when the number of initiated broadcasts drastically increases.

The experiment consists in simulating 1000 processes that have a probability to broadcast messages that varies over time. In this experiment, the *maxLatency* parameter is set to 10 and there is no churn. Moreover, the warm-up phase is not represented for the sake of clarity. Figure 10 plots both the average number of sequence number (SN) requests received by each coalition member at the start of each round (first Y axis) and the average number of broadcasts initiated at the start of each round (second Y axis). Each “coalition X” plot depicts the life cycle of a coalition (i.e. the cycle at which it is created/destroyed) and the average number of SN requests received by each member.

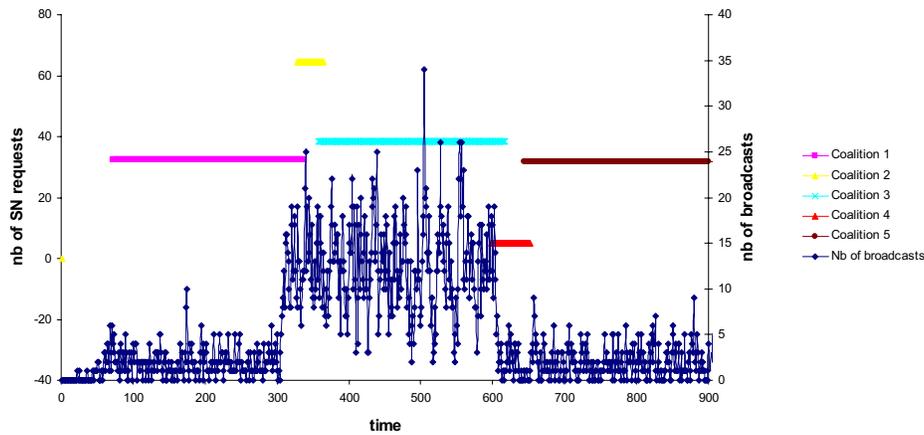


Fig. 10. Self-sizing mechanism.

The self-sizing mechanism was parameterized to maintain the average number of SN requests by member between 30 and 40. From cycle 0 to cycle 300, processes have a low probability to initiate a new broadcast at the start of each round. During this period, messages are sequenced by coalition 1, which contains 3 members that handle (on average) 32,5 SN requests per cycle. Then, the broadcast rate significantly increases between cycles 300 and 600. Coalition 1 is first replaced by coalition 2 that contains 6 members handling 64,7 SN requests per cycle on average. Thus coalition 2 does not yet have enough members to handle the load. Consequently, coalition 2 is replaced by coalition 3 after a short period of time. The latter contains 12 members handling 38,5 SN requests per cycle on average. At time 600, the broadcast rate suddenly decreases. Coalition 3 is first replaced by coalition 4 (7 members and 15 SN requests per cycle), and then by coalition 5 (4 members and 31,3 SN requests per cycle).

This experiment shows that the self-sizing mechanism allows keeping the average number of SN requests within the desired range, thus ensuring that coalitions can sustain a constant throughput, regardless of the broadcast rate.

5 Related Work

Update ordering for eventual consistency can be ensured by using total order protocols like the ones described in [6]. However only optimistic total order protocols can efficiently support eventual consistency in a large scale setting [21,18]. Other approaches to total ordering are too strong and would decrease responsiveness.

Optimistic total order protocols distinguish between tentative delivery and committed delivery of messages. This approach has been proposed by Kemme et al. in [13] to improve the responsiveness of the system in a LAN. The optimistic approach in this case is based on the spontaneous total ordering in LANs. The protocol proposed by Vincente and Rodrigues in [21,18] guarantees that the tentative order is equal to the committed one during synchrony periods of the network. During periods of asynchrony rollbacks might occur. Finally, the protocol proposed by Sousa et al in [18] does its best to guarantee that the tentative order is equal to the committed by artificially delaying messages received at a process before delivery through a mechanism called delay compensation. This delay based approach aims at creating the right conditions for spontaneous total ordering in WANs.

Note that optimistic total order protocols deterministically guarantee eventual consistency by relying on strong reliable update dissemination. However, reliable information dissemination does not scale and cannot be employed in weakly connected environments. This is why systems and protocols which target those environments use epidemic dissemination, thus providing eventual consistency with high probability [19,15,16,2].

For instance, Bayou [19] is a storage system designed for a weakly connected computing environment. In Bayou, one server, designated as the primary, takes responsibility for totally ordering updates and thus for deciding the committed order. Each secondary replica executes updates in a tentative order while the committed order is being decided. Update propagation follows an anti-entropy [7] mechanism: pairs of replicas periodically exchange information to update their states. This pair-wise communication copes with arbitrary network connectivity and after an arbitrary number of communication exchanges, replicas converge to an identical state.

Oceanstore [15] targets extremely wide distributed environments with huge numbers of users. Contrarily to our protocol, in Oceanstore, consistency is achieved in a *conscious* manner. Indeed, consistency is reached using a two-tier architecture: a specific small set of untrusted servers, called the inner ring of the object, store the primary object replicas (primary tier). Other replicas, called secondaries, are deployed on a large number of nodes, mostly for caching reasons (secondary tier). The inner-ring totally orders updates coming from any node

hosting a replica using a Byzantine agreement protocol [5]. To guarantee the termination of the Byzantine agreement, inner-ring servers form a sub-network with high bandwidth and a fixed infrastructure. Updates are disseminated epidemically. In particular, tentative updates are pushed through secondary replicas. The committed updates dissemination follows two steps: (1) a best-effort multicast using the dissemination tree connecting the primary-tier to the secondary-tier and (2), secondary replicas pull missing information from parents and the primary tier.

6 Concluding Remarks

This paper introduces the notion of *unconscious* eventual consistency. In contrast to conscious eventual consistency, it can be implemented in permanently asynchronous environments, while still supporting important classes of distributed applications such as interactive applications based on continuous shared data. The paper describes a protocol that implements unconscious eventual consistency based on gossiping. The protocol is stable, non-blocking, and scalable. Our simulations convey the reasonable latency of the protocol during synchronous periods, and its high throughput under load.

References

1. PeerSim: A Peer-to-Peer Simulator, 2006. <http://peersim.sourceforge.net/>.
2. Karl Aberer, Magdalena Puceva, Manfred Hauswirth, and Roman Schmidt. Improving data access in p2p systems. *IEEE Internet Computing*, 6(1):58–67, 2002.
3. Sumeer Bhola and Mustaque Ahmad. 1/k phase stamping for continuous shared data (extended abstract). In *PODC '00: Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 181–190, New York, NY, USA, 2000. ACM Press.
4. Kenneth P. Birman, Mark Hayden, Ozgur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, 1999.
5. Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.
6. Xavier Defago, Andre Schiper, and Peter Urban. Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey. *ACM Comput. Surv.*, 36(4):372–421, 2004.
7. Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM Press.
8. P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight Probabilistic Broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.
9. M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, 1985.

10. K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, October 2003.
11. I. Gupta, K. Birman, and R. van Renesse. Fighting fire with fire: using randomized gossip to combat stochastic scalability limits. *Journal of Quality and Reliability Engineering International*, 2002.
12. Indranil Gupta, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *SRDS*, pages 180–189. IEEE Computer Society, 2002.
13. B. Kemme, F. Pedone, G. Alonso, and A. Schiper. Processing Transactions over Optimistic Atomic Broadcast Protocols. In *Proceedings of 19th International Conference on Distributed Computing Systems (ICDCS'99)*, 1999.
14. Anne-Marie Kermarrec, Laurent Massoulié, and Ayalvadi J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. Parallel Distrib. Syst.*, 14(3):248–258, 2003.
15. John Kubiawicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*, November 2000.
16. Y. Saito, C. Karamanolis, M. Karlsson, and M. Mahalingam. Taming Aggressive Replication in the Pangaea Wide-area File System. *ACM SIGOPS Operating Systems Review*, 36, 2002.
17. Yasushi Saito and Marc Shapiro. Optimistic Replication. *ACM Computing Survey*, 37(1):42–81, 2005.
18. Antonio Sousa, Jos Pereira, Francisco Moura, and Rui Oliveira. Optimistic Total Order in Wide Area Networks. In *21st IEEE Symposium on Reliable Distributed Systems*, pages 190–199. IEEE CS, October 2002.
19. D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 172–182, New York, NY, USA, 1995. ACM Press.
20. IBM T.J.-Watson. Online gaming. <http://www.research.ibm.com/gaming>.
21. Pedro Vicente and Luís Rodrigues. An Indulgent Uniform Total Order Algorithm with Optimistic Delivery. In *21st Symposium on Reliable Distributed Systems (SRDS 2002)*, pages 92–101, Osaka, Japan, 2002.

Improving Byzantine Protocols with Secure Computational Components*

Miguel Correia[†] Alysson N. Bessani[‡] Nuno F. Neves[†] Lau C. Lung[§] Paulo Veríssimo[†]

[†] Faculdade de Ciências da Universidade de Lisboa, Bloco C6, Campo Grande, 1749-016 Lisboa – Portugal

[‡] Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina – Brazil

[§] Programa de Pós-Graduação em Informática Aplicada, Pontifícia Universidade Católica do Paraná – Brazil

Abstract

Byzantine-tolerant protocols are currently being used as building blocks for the construction of secure applications, which means that a better understanding of their performance characteristics can have an important practical impact. The paper investigates if these protocols can benefit performance-wise with the inclusion in the nodes of a local secure computational component. Using this extended model, the paper describes two simple protocols that solve respectively the multi-valued and vector consensus problems. Then, it compares the behavior of the two protocols with several other similar protocols from the literature, but for systems without secure components, in terms of the achieved resilience, time and communication complexities. This analysis shows that the two novel protocols have the best overall performance, which indicates that the proposed extended architecture can be an attractive solution for some environments.

1 Introduction

The development of efficient distributed protocols has both theoretical and practical interest. Today, Byzantine-tolerant protocols are being used as important building blocks for the construction of secure applications based on a recent approach: *intrusion tolerance* [1, 18, 36]. This approach can be considered to be part of the ongoing effort to make computing systems more secure, Internet included, vis-a-vis the large number of security incidents permanently reported by entities like CERT/CC¹.

*This work was partially supported by the EU through NoE IST-4-026764-NOE (RESIST) and project IST-4-027513-STP (CRUTIAL), and by the FCT through project POSI/EIA/60334/2004 (RITAS) and the Large-Scale Informatic Systems Laboratory (LASIGE).

¹<http://www.cert.org/stats>

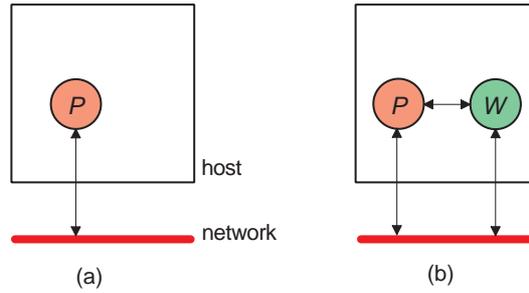


Figure 1: (a) Typical system architecture. (b) System architecture explored in the paper.

Research in message-passing distributed protocols typically considers a set of nodes interconnected by a network, each one running a software component called a process (see Figure 1(a)). Works in this area that aim intrusion tolerance have to assume that the system can suffer from Byzantine faults, including malicious attacks [3–7, 11, 13–15, 25, 27, 30]. Therefore, processes can fail in an unconstrained manner (i.e., they can violate the protocol specification in any possible way) and the network may corrupt the communication, e.g., by dropping, modifying or repeating messages. Regarding timeliness, the system is usually considered to be mostly asynchronous (to avoid time assumptions that could be violated, e.g., through denial of service attacks). However, in order to circumvent the FLP impossibility result [17], the system usually has to be extended with some oracle [3, 13, 14, 25] or time assumptions [15]. Another approach that has been used to circumvent this result, and that we will use in the paper, is randomization [4–7, 9, 27, 30].

The paper investigates the benefits for the performance of Byzantine-tolerant protocols of making this picture slightly more complicated. Suppose each node now includes a second component w that can communicate both with the process p (locally) and with similar components in other nodes (through the network). The extended architecture, which includes the *wormhole* component w , is shown in Figure 1(b). Notice that nothing is being said at this stage about what w is – it might be either a hardware or a software module. However, w has the important characteristic that it can only fail by crashing (fail-stop), not in a Byzantine way, i.e., it is secure or tamperproof. Therefore, this assumption implies a *hybrid fault model* where nodes have a part that can only fail by crashing (w) and the rest that can fail arbitrarily (everything except w)². Each of these components (wormholes) includes a random oracle, i.e., a generator of random bits. We do not consider any other oracles or time assumptions.

Two concerns may reasonably be raised about this model. First: is it possible to implement such a

²This kind of fault model is clearly different from some previous work in hybrid fault models, starting in [26], in which fault distributions are simply assumed. Here we design the secure component with the purpose of enforcing its fault model, an obvious requirement in environments prone to malicious faults.

model or is it of purely theoretical interest? The answer is simple: there are many ways of implementing w . Most computers, either laptops, desktops or servers can be extended with some kind of secure hardware that can be used to run w , e.g., USB tokens, smartcards, secure coprocessors and PC/104 appliances³. Software solutions include running w as a special process in a security kernel (e.g., Perseus [32]).

The second question is: is this model relevant? Why cannot we just make p secure, i.e., $p = w$, and end up with a fail-stop model? The first answer is that sometimes it is possible to do so and a system designer should consider that possibility. However, in general p is part of a complex application with millions of lines of code that cannot be put inside w (USB tokens and smartcards have very limited resources) and cannot even be secured in that way because it has complex interactions with the environment, e.g., with human users, networks and files. If we want to make w secure or fail-stop, it has to satisfy two properties derived from the classical reference monitor properties [20]⁴:

- Isolation. The wormhole must be tamperproof or secure. This is considered to be an assumption throughout the paper, although it has to be enforced in an implementation.
- Verifiability. Its security has to be formally verifiable. This is true for the instances of w we present in the paper, since they implement reasonably simple distributed protocols.

The issue explored in the paper is: what are the benefits for the performance of Byzantine-tolerant protocols of the model in Figure 1(b)? Is there any interest for such protocols of having a secure component in the nodes?

Context and related work. We have been exploring this kind of hybrid fault models by calling these subsystems *wormholes* [34]. The metaphor comes from an astrophysics concept that some Science Fiction has presented as shortcuts that might be used to travel fast to faraway places in the Universe⁵. The idea we have been exploring is to take advantage of components with stronger properties to handle some kind of uncertainty. The first work in this line used a distributed real-time wormhole to handle uncertainty in terms of time [35]. Afterwards, a distributed real-time and secure wormhole was used to build Byzantine fault-tolerant protocols, i.e., to handle uncertainty in terms of malicious faults [10, 12, 29].

³Several cryptographic modules that might be used with this purpose were validated for conformance to FIPS PUB 140-1 and FIPS PUB 140-2 (Security Requirements for Cryptographic Modules) by the National Institute of Standards and Technology. A list is available at: <http://csrc.nist.gov/cryptval/140-1/1401val.htm>.

⁴The third property that a reference monitor must satisfy is specific for access control (completeness).

⁵See, e.g., <http://en.wikipedia.org/wiki/Wormhole>

The present paper follows this work on wormholes but has an important difference. Our purpose here is to compare protocols with/without wormholes that are as comparable as possible. More specifically, we propose protocols as similar as possible to protocols that do not use wormholes, and we do not make more time assumptions about the wormholes than about the rest of the system. The approach used in this paper is to consider only randomized protocols since they do not rely on any time assumptions, so we make no time assumptions about any part of the system or wormholes, i.e., the wormholes are asynchronous. This is an interesting property since the wormholes can use the same network as the processes to communicate without being vulnerable to attacks against time assumptions.

Several security protocols have been previously proposed that use different types of secure components to prevent intrusions in critical modules. Tygar and Yee showed how a secure coprocessor can be used, e.g., to guarantee the security of an electronic payment scheme [33]. Itoi and Honeyman used smartcards for secure authentication with Kerberos [22]. Shoup and Rubin used also smartcards to enhance the security of session key distribution [31]. Avoine and colleagues presented an algorithm for deterministic fair exchange also based on secure components [2]. Several other examples might be listed. However, all these works use secure components with the purpose of ensuring some of the security properties of the protocols. To the best of our knowledge this paper is the first that uses secure components with the purpose of *improving the performance of distributed systems algorithms* like consensus⁶. Here the purpose is not to prevent intrusions in certain components and ensure certain security properties, but to improve the performance of protocols that tolerate intrusions in some of the nodes.

Paper results. The contributions of the paper are the following:

- it presents the first work with strictly asynchronous wormholes, using randomization to circumvent the FLP impossibility; it also uses I/O automata to formalize a system with wormholes.
- it compares consensus protocols with and without wormholes as similar as possible, showing the benefits of using these secure components; protocols for multi-valued and vector consensus are also provided;

⁶There are secure hardware modules that are slow when compared to servers or even common PCs. If the wormholes were implemented using that hardware, the performance of the protocols might become worse instead of better. We assume that the wormholes are PC/104 appliances or other fast hardware modules, whose overhead would be low when compared with the communication delay.

2 System Model

The system considered in the paper has a set of n processes $\Pi = \{p_1, p_2, \dots, p_n\}$ and a set of n wormholes $\Upsilon = \{w_1, w_2, \dots, w_n\}$. A node i contains a process p_i that can access the local wormhole w_i (see Figure 1(b)). Each wormhole includes a random oracle module. This oracle provides random numbers uniformly distributed from a finite set \mathcal{U} . We postpone the discussion about the content of \mathcal{U} to Section 2.2. The system is asynchronous, i.e., no bounds are assumed on processing and communication delays.

We will model the system components using I/O automata, a formalism introduced by Lynch and Tuttle [23, 24]. An automaton receives input actions and generates output and internal actions. A system is represented by a composition of automata.

2.1 Fault Model

The architecture we are considering is more complex than what is commonly considered in message-passing distributed algorithms so there is also more to be said about the fault model.

A process is said to be *correct* if it does not *fail* during the execution of the protocol, i.e., if it follows the protocol. Otherwise, it is said to be *corrupt* or *failed*. Processes can fail in the usual Byzantine ways, for instance: they can stop, delay the communication, send spurious messages, or transmit several messages with the same identifier. Corrupt processes can also pursue a plan of breaking the properties of the protocol alone or in collusion with other failed processes. We use the letter f to denote the maximum number of processes that are allowed to fail during an execution of the protocol.

The extended architecture also introduces new failure modes. A wormhole is said to be *correct* if it does not fail, i.e., if it does not *crash*. Otherwise it is said to be *crashed* or *failed*. In addition to the situations listed above, a process p_i can fail if w_i crashes, if p_i does not manage to communicate with w_i (e.g., because an attacker controls the node) or if its communication with w_i is modified in some way.

Since our protocols do not require processes to communicate directly, but only through the wormholes, we will not make any statement about assumptions on the network channels of the processes. The wormholes are fully-connected by private channels with three properties: if the sender and the recipient of a message are both correct then (1) the message is eventually received; (2) the message is not modified in the channel; and (3) the content of the message can only be disclosed by the sender or the recipient. In other words, the communication can be delayed arbitrarily, but all messages are eventually delivered correctly. In practice, these channels can be implemented in common LANs or the Internet using secure

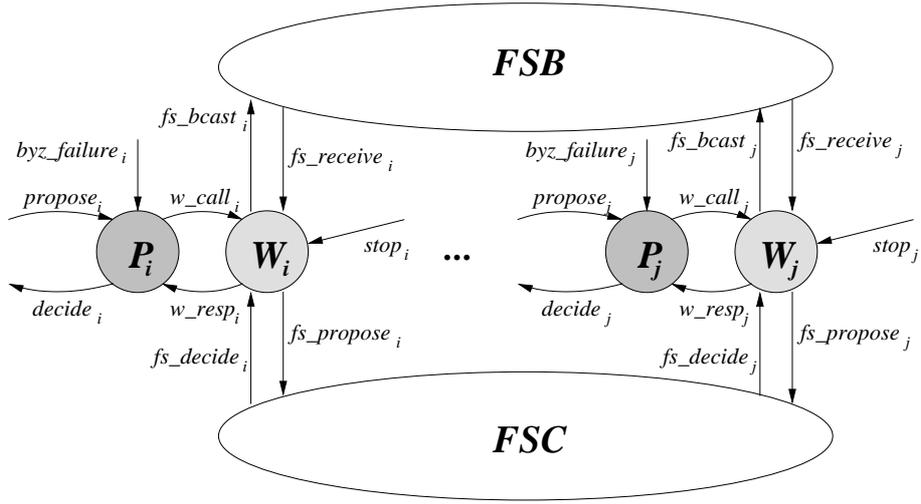


Figure 2: Automata composition for all protocols in the paper. The automaton p_i models process p_i , automaton w_i models wormhole w_i , automaton FSB a fail-stop broadcast, and FSC a fail-stop multi-valued consensus.

communication protocols such as the Secure Socket Layer [19]. Notice that the assumption of private channels is a way of hiding (masking) the failures in the network, such as message modifications, replays, omissions and spurious messages.

2.2 Formal System Model

Each process p_i is modeled as an automaton with five actions (see Figure 2):

- input $propose(v)_i$ – invocation of the protocol by user U_i ;
- output $decide(v)_i$ – response to user U_i with the value decided by the protocol;
- output $w_call(v)_i$ – passage of a value to the wormhole w_i ;
- input $w_resp(v)_i$ – response from the wormhole w_i ;
- input $byz_failure_i$ – signals the Byzantine failure of the automaton.

The user U_i corresponds to the software component that calls process p_i . The automata composition in the figure represents the system that executes the protocols, therefore it does not include that user. However, U_i might also be modeled as an automaton.

Wormhole w_i is modeled as an automaton with several actions, some of which are (see the same figure):

- input $w_call(v)_i$ – corresponds to the output with the same name in process p_i ;
- output $w_resp(v)_i$ – corresponds to the input with the same name in process p_i ;
- input $stop_i$ – signals the crash of the wormhole.

The composition includes a broadcast channel FSB with a semantics equivalent to the sender wormhole individually sending the message to all wormholes in Υ (including itself) using the private channels presented in the previous section. The primitive is used by the wormholes that are fail-stop, therefore all recipients receive the same message unless a wormhole crashes. The signature is the following:

- input $fs_bcast(v)_i$ – send the value v to every wormhole $w_j \in \Upsilon$;
- output $fs_receive(j, v)_i$ – receive a value v from process j .

This signature includes two actions related to the fail-stop multi-valued consensus automaton FSC , which we describe in the next section: $fs_propose(v)_i$ and $fs_decide(v)_i$.

The failures of wormholes (crash) and processes (Byzantine) are modeled by inputs with distinct meanings. The input $stop_i$ is the usual way of modeling the crash of an automaton [23] and is handled explicitly in the code of the automaton (see Automaton 2). The use of an input $byz_failure_i$ to model Byzantine failures was first suggested in [8]. When this event occurs, the automaton is substituted by another automaton with the same signature but with arbitrary behavior.

Consensus Among Wormholes

The protocols presented in the paper use as building block a multi-valued consensus protocol executed by the wormholes. This protocol does not need to tolerate Byzantine faults since the wormholes are assumed to be fail-stop and fully-connected by private channels (Section 2.1), which means that the protocol is in essence a *fail-stop consensus*. In order to circumvent the FLP result, the protocol resorts to the above mentioned random oracle modules.

The fail-stop consensus is modeled as a single automaton FSC (Figure 2). In terms of system architecture, this automaton models part of the behavior of the wormholes (the fail-stop consensus) plus the private channels connecting the wormholes. Therefore, in reality, any wormhole w_i is modeled by the automata w_i and FSC. The objective of modeling the fail-stop consensus as a separate automaton was to attain modularity, thus allowing us to plug-in different consensus modules into the wormholes.

Informally, consensus is the problem of making a set of entities (processes, wormholes) agree on a common value. A wormhole w_i is said to *propose* a value $v \in \mathcal{V}$ for an execution of the consensus

protocol when an output action $fs_propose(v)_i$ occurs in w_i . The wormhole is said to *decide* on a value v when an input action $fs_decide(v)_i$ occurs in w_i . Consensus is defined in terms of the following properties:

- *Validity-1*: If a correct wormhole decides v , then v was proposed by some wormhole.
- *Agreement*: No two correct wormholes decide differently.
- *Termination*: Every correct wormhole eventually decides with probability 1.

In the paper we use two variants of consensus protocols: binary consensus ($\mathcal{V} \equiv \{0, 1\}$) and multi-valued consensus (\mathcal{V} is a finite set of values). An example of a fail-stop binary consensus protocol is presented in [4], while a fail-stop multi-valued consensus can be found in [16]. The random oracle used in the former provides values in the set $\mathcal{U} \equiv \{0, 1\}$, while in the latter provides values in $\mathcal{U} \equiv \{1, 2, \dots, n\}$ (where n is the total number of processes). A transformation from binary to multi-valued fail-stop consensus is presented in [28]. All these protocols tolerate the failure of at most half less one processes/wormholes ($f = \lfloor \frac{n-1}{2} \rfloor$).

3 Byzantine Consensus

Consensus is an important distributed systems problem since it can be used as the main building block to solve several other agreement problems [11, 21]. Several protocols for Byzantine consensus in asynchronous systems have been proposed, using several methods to circumvent FLP: randomization [4, 30], failure detectors [3, 25], partial-synchrony [15] and distributed wormholes [10].

The *resilience* of a distributed protocol is the maximum number of failed processes it can tolerate. The maximum resilience for Byzantine consensus in asynchronous systems is $f = \lfloor \frac{n-1}{3} \rfloor$ out of a total of n processes [5, 11, 15], which is also the resilience of the protocols we propose in the paper.

3.1 Multi-Valued Consensus

The specification of *multi-valued consensus* for a system with Byzantine faults is similar to the definition given in Section 2.2, but has a different Validity property [10, 15, 25]:

- *Validity-2*. If all correct processes propose the same value v , then any correct process that decides, decides v .
- *Agreement*: No two correct processes decide differently.

Automaton 1 Consensus protocol (process p_i)

Signature:Input: $propose(v)_i, w_resp(V)_i, byz_failure_i$ Output: $w_call(v)_i, decide(v)_i, decide(\perp)_i$ $v \in \mathcal{V}, \perp \notin \mathcal{V}$ **State:** $prop = \perp$, value proposed by the user $Vect = \perp$, vector with several proposed values $failed = false$, true if the process is corrupt**Transitions:**1: input $propose(v)_i$ 2: Eff: $prop \leftarrow v$ 3: input $w_resp(V)_i$ 4: Eff: $Vect \leftarrow V$ 5: input $byz_failure_i$ 6: Eff: $failed \leftarrow true$ 7: output $w_call(v)_i$ 8: Pre: $prop = v$ 9: Eff: $prop \leftarrow \perp$ 10: output $decide(v)_i$ 11: Pre: $\#_v(Vect) \geq f + 1$ 12: Eff: $Vect \leftarrow v$ 13: output $decide(\perp)_i$ 14: Pre: $\forall v, \#_v(Vect) < f + 1$ 15: Eff: $Vect \leftarrow \perp$

- *Termination*: Every correct process eventually decides with probability 1.

Our protocol solves *multi-valued consensus* if FSC is instantiated with a multi-valued fail-stop consensus. A direct consequence of the system architecture depicted in Figure 1(b) is that the protocol is run both in the processes and the wormholes (respectively p and w). Automata 1 and 2 correspond respectively to the code of the processes and wormholes. Recall that our objective is to compare protocols that are as similar as possible. This is why the automata are very simple and leave most of the work to the FSC.

The presentation of the protocol assumes a property of well-formedness, both for the users that call the protocol (U_i) and for the processes (p_i) [23]:

- *Well-formedness*. For any i , the interactions between U_i and p_i , and the interactions between p_i and w_i , are *well-formed* for i .

Let us consider the interaction between the user U_i and the automaton p_i . A sequence of actions $propose(v)_i$ and $decide(v)_i$ is said to be *well-formed* for i if it is some prefix of the cyclically ordered sequence $propose(v')_i, decide(v'')_i, propose(v''')_i, decide(v''''_i), \dots$ This property essentially excludes the possibility of a user making two proposals before the decision of the protocol is returned. The objective of making this assumption is to make Automaton 1 simpler, by not having to consider explicitly

Automaton 2 Consensus protocol (wormhole w_i)

Signature:

Input: $w_call(v)_i, fs_receive(j, v)_i, fs_decide(v)_i,$
 $stop_i$

Output: $fs_propose(v)_i, fs_bcast(v)_i, w_resp(v)_i$
 $v \in \mathcal{V}, \perp \notin \mathcal{V}$

State:

$prop = \perp$, value proposed by the process

$dec = \perp$, vector decided by FSC

$\forall j \in \Pi : Vect[j] = \perp$, vector with values delivered by
FSB

$stopped = false$, true if the wormhole is stopped

Transitions:

1: input $w_call(v)_i$

2: Eff: $prop \leftarrow v$

3: input $fs_receive(j, v)_i$

4: Eff: $Vect[j] \leftarrow v$

5: input $fs_decide(Vect)_i$

6: Eff: $dec \leftarrow Vect$

7: input $stop_i$

8: Eff: $stopped \leftarrow true$

9: output $fs_bcast(v)_i$

10: Pre: $\neg stopped \wedge prop = v$

11: Eff: $prop \leftarrow \perp$

12: output $fs_propose(Vect)_i$

13: Pre: $\neg stopped \wedge \#_{\perp}(Vect) \leq f$

14: Eff: $\forall j \in \Pi : Vect \leftarrow \perp$

15: output $w_resp(Vect)_i$

16: Pre: $\neg stopped \wedge dec = Vect$

17: Eff: $dec \leftarrow \perp$

the case of ill-formed interactions. Nevertheless, this assumption might be discarded with simple modifications to the algorithm, like identifying each consensus execution with a consensus identifier (cid), and substituting the variables $prop$ and dec by sets containing tuples (cid, v) for the active consensuses. Similar considerations might be done about the well-formedness of the interactions between p_i and w_i .

The protocol follows the typical structure of I/O automata protocols [23, 24]. Each automaton starts with the declaration of its signature, i.e., its input and output actions. Then, it declares the state variables and the transitions corresponding to each action, specified in terms of preconditions ($Pre:$) and effects ($Eff:$).

In the protocol, vectors have one entry per process in Π (or wormhole in Υ) and are designated by an uppercase letter. Function $\#_x(Vect)$ counts the number of occurrences of x in vector $Vect$. In Automaton 1, line 11, this function is used to select a value v that occurs at least $f + 1$ times in $Vect$. If there are two values v_1 and v_2 in that condition, the function returns the one that appears first in $Vect$.

The protocol works as follows. It starts with an action $propose(v)_i$ in each process p_i , which sets $prop = v$ (Aut.1:1-2)⁷, and causes an output action $w_call(v)_i$ (Aut.1:7-9). This output action

⁷We use this abbreviated notation to reference Automaton 1, lines 1 to 2.

corresponds to the input action with the same name in the wormhole automaton (recall Figure 2). In other words, an action $propose(v)_i$ causes the value v to be passed to the wormhole w_i . Then, w_i broadcasts v to all wormholes, including itself (Aut.2:1-2,9-11). When w_i receives a value, it puts it in a vector $Vect$ (Aut.2:3-4). When w_i has f or less empty values in $Vect$, it proposes the vector to the fail-stop consensus (Aut.2:12-14). When this protocol terminates, w_i returns the vector decided to the process (Aut.2:5-6,15-17). If a value appears $f + 1$ times in the vector, or more, this is the value decided to guarantee Validity-2, otherwise a default value is decided (Aut.1:10-15).

The correctness proof of the following Theorem is in Appendix A:

Theorem 1 *If at most $f = \lfloor \frac{n-1}{3} \rfloor$ processes are failed, then the protocol specified by Automata 1 and 2 solves consensus as specified by properties Validity-2, Agreement and Termination.*

3.2 Vector Consensus

Vector consensus is a variant of the problem of consensus, which is specially interesting when Byzantine faults are considered [3, 14, 29]. A vector consensus protocol instead of deciding a value, returns a vector. This vector has values proposed by a majority of correct processes, something that can be useful to solve practical distributed system problems, like atomic multicast [11]. The problem is a variation of *interactive consistency* for asynchronous systems, since in these systems it is not possible to guarantee that the vector has values from all correct processes. The definition is the same as for the consensus protocols above, except for the Validity property:

- *Validity-VC*: Every correct process decides on a vector $Vect$ of size n , such that:
 1. For every $1 \leq i \leq n$, if process p_i is correct, then $Vect[i]$ is either the initial value of p_i or the value \perp , and
 2. at least $f + 1$ elements of the vector $Vect$ are the initial values of correct processes.
- *Agreement*: No two correct wormholes decide differently.
- *Termination*: Every correct wormhole eventually decides with probability 1.

A protocol that solves vector consensus is presented in Automata 3 (process automaton) and 2 (wormhole automaton, same as for binary/multi-valued consensus). The protocol is a straightforward modification of the previous multi-valued consensus. It simply returns the vector decided by the wormholes, instead of choosing the most frequent value in the vector. The automaton FSC has also to be instantiated with a multi-valued fail-stop consensus.

Automaton 3 Vector consensus protocol (process p_i)

Everything identical to Algorithm 1 except lines 10-15 that are substituted by:

- 10: output $decide(Vect)_i$
11: Pre: $Vect \neq \perp$
12: Eff: $Vect \leftarrow \perp$
-

Theorem 2 *If at most $f = \lfloor \frac{n-1}{3} \rfloor$ processes are failed, then the protocol specified by Automata 3 and 2 solves vector consensus as specified by properties Validity-VC, Agreement and Termination.*

The correctness proof of the Theorem is in Appendix A.

4 Evaluation of the Protocols

Randomized Byzantine agreement protocols are usually evaluated in terms of resilience, time and communication complexities [9]. *Time complexity* in asynchronous systems is normally measured by counting the number of asynchronous rounds. In this kind of protocols, an asynchronous round is defined in the following way: a process broadcasts a message to all other processes, and then waits for $(n - f)$ messages broadcasted by the others in the same round; when it gets that number of replies, it either goes to the next round or terminates. For randomized protocols, the metric is usually the *expected* number of asynchronous rounds, since the number of rounds can only be defined probabilistically. We evaluate the protocols in the situation where the failed processes do the best they can to delay the protocol and, for multi-valued consensus, all correct processes have different initial values. *Communication complexity* can be measured in number of bits sent (per round or protocol execution) or in number of transmitted messages. Here we use the expected number of message broadcasts.

Table 1 compares our protocols with several other asynchronous randomized consensus protocols that have been published previously. Each row has information about one protocol. The columns have the obvious meanings except the last that contains a reference to the place where the algorithm is described: a section of this paper (e.g., §3 denotes Section 3 and §B denotes Appendix B), another paper (e.g., [4]), or a combination of more than one of those other types of references (indicated by the addition operator).

The top rows (1-5) of the table evaluate fail-stop consensus protocols, the middle rows (6-7) evaluate Byzantine consensus, and the bottom rows (8-9) our own protocols. Our protocols need a multi-valued fail-stop consensus for the implementation of the FSC (Section 2.2). The fail-stop consensus protocol we use is a combination of a binary consensus plus the ‘binary to multi-valued’ transformation in [28], since

	Consensus type	Fault model	Oracle	Resilience	Expected Time complexity	Expected Communication complexity	Reference
1	binary	crash	random	$\lfloor \frac{n-1}{2} \rfloor$	$2^{n-1} + 1$	$(2^{n-1} + 1)n$	[4]
2	binary	crash	random	$\lfloor \frac{n-1}{3} \rfloor$	$1.5 \times 2^{n-f-1} + 2.5$	$(1.5 \times 2^{n-f-1} + 2.5)n$	§B
3	multi-val.	crash	random	$\lfloor \frac{n-1}{2} \rfloor$	$2^{n-1} + 2$	$(2^{n-1} + 1)n + n^2$	[4]+[28]
4	multi-val.	crash	random	$\lfloor \frac{n-1}{3} \rfloor$	$1.5 \times 2^{n-f-1} + 3.5$	$(1.5 \times 2^{n-f-1} + 2.5)n + n^2$	§B+[28]
5	multi-val.	crash	random	$\lfloor \frac{n-1}{2} \rfloor$	$n^{n-1} + 2$	$(n^{n-1} + 1)n + 2n^2$	[16]
6	binary	Byz.	random	$\lfloor \frac{n-1}{5} \rfloor$	$2^{n-f-1} + 1$	$(2^{n-f-1} + 1)n$	[4]
7	binary	Byz.	random	$\lfloor \frac{n-1}{3} \rfloor$	$4.5(2^{n-f-1} + 1)$	$(2^{n-f-1} + 1)3n^2$	[5]
8	multi-val.	Byz.	wormh.	$\lfloor \frac{n-1}{3} \rfloor$	$1.5 \times 2^{n-f-1} + 4.5$	$(1.5 \times 2^{n-f-1} + 4.5)n + n^2$	§3+§B+[28]
9	vector	Byz.	wormh.	$\lfloor \frac{n-1}{3} \rfloor$	$1.5 \times 2^{n-f-1} + 4.5$	$(1.5 \times 2^{n-f-1} + 4.5)n + n^2$	§3.2+§B+[28]

Table 1: Comparison of several asynchronous randomized consensus protocols.

the only randomized (crash-tolerant) multi-valued consensus in the literature is quite inefficient (row 5) [16]. For reasons we explain below, instead of the binary protocol in [4] (row 1) we use a modified – fail-stop – version of the Byzantine protocol in [5], which is presented in Appendix B.

The first conclusion we take from the table is that the time and message complexities of our protocols are similar to the best complexities of fail-stop protocols. Even though, we do not describe a binary consensus protocol, notice that a multi-valued consensus also solves the binary consensus if we set $\mathcal{V} = \{0, 1\}$. Comparing the binary consensus in rows 2 and the ‘binary’ consensus in row 8 (i.e., the multi-valued consensus with $\mathcal{V} = \{0, 1\}$) and the multi-valued consensus in rows 4 and 8, we get to that conclusion that the time/message complexities of our protocols are similar to the best of fail-stop protocols.

The second conclusion is that these complexities are better than those of Byzantine resilient protocols in the literature. Comparing our ‘binary’ protocol (row 8) to Bracha’s protocol [5] (row 7) we see that the time complexities are both $O(2^{n-f})$ (although our multiplying constant is slightly lower) but our communication complexity is clearly lower: $O(2^{n-f}n)$ against $O(2^{n-f}n^2)$. The time and communication complexities of [4] (row 6) are apparently equal to ours, respectively $O(2^{n-f})$ and $O(2^{n-f}n)$. However, the resilience of that protocol is suboptimal (only $\lfloor \frac{n-1}{5} \rfloor$ out of n) so if we wanted to tolerate the same number of faults the complexities would be considerably worse (exponential with a base greater than 2). We did not find multi-valued or vector consensus of the class we are considering in the literature, so we cannot make a comparison for those protocols.

The reason why we used a modified version of Bracha’s protocol instead of Ben-Or’s protocol [4] to

evaluate our protocols can now be understood. Ben-Or’s protocol tolerates $f = \lfloor \frac{n-1}{2} \rfloor$ crashes (row 1), the optimal resilience for fail-stop protocols. The time and communication complexities are respectively $O(2^n)$ and $O(2^{n-1})$. The resilience of the protocol is more than we need for our binary consensus protocol, but its complexities would lead to similar complexities for our protocols. The problem is that a time complexity of $O(2^n)$ is worse than the complexities of current Byzantine-resilient protocols (rows 6-7). The same is true for the communication complexity. Using the modified version of Bracha’s protocol (Appendix B) we manage to have better complexities: $O(2^{n-f})$ and $O(2^{n-f-1})$. The resilience of this modified protocol is suboptimal for fail-stop protocols but exactly what we need since it is used to support a Byzantine protocol.

On the kind of protocols compared. Randomized consensus protocols are essentially of two kinds:

1. Those based on local random oracle modules, following Ben-Or’s seminal paper [4];
2. Those based on a shared coin that provides identical random numbers to all processes, starting with Rabin’s work [30].

In the comparison and table below, we consider only protocols of the first kind because we are interested in comparing protocols as similar as possible and there is nothing similar to Rabin’s type of protocols for the fail-stop model.

Protocols of the second type typically have lower time complexities than protocols of the first. There are even protocols of the second type that run in a constant expected number of rounds, while protocols of the first have exponential expected time complexities. However, our point here is to compare protocols with wormholes with protocols based on the usual architecture, so we do not want to use shared coins. Moreover, some of us have recently shown that protocols of the second type run in a low number of rounds under realistic faultloads, including Byzantine nodes [27].

5 Conclusion

The need for more secure distributed systems is raising a renewed interest in efficient Byzantine protocols. This paper investigates the contribution for that objective of including a secure component – wormhole – inside the system nodes. The paper compares a set of Byzantine protocols based on the typical model (nodes interconnected by a network) with our model. For this comparison to make sense we consider randomized protocols, only with local random oracles (i.e., following Ben-Or [4]), which have

been recently shown to be efficient under realistic faultloads. The conclusion from that comparison is that our approach manages to reduce the complexities of Byzantine protocols to complexities equivalent to fail-stop protocols, which are considerably better than those of previous similar Byzantine protocols.

The protocols proposed are quite simple and most of the functionality ended up being put in the wormholes. This seems to contradict our comment in the introduction that it is not possible to make $p = w$, i.e., to put everything inside the wormholes. However, the contradiction does not exist because consensus is only a component to be used in larger applications with complex interactions with users, networks and files. Moreover recall that the reason why we put most of the functionality inside the wormholes was to have similar protocols with and without wormholes, something that is no clear how could be done if the functionality was divided between processes and wormholes, e.g., like in [10, 29].

The paper follows the line of research in systems extended with wormholes, but considers, for the first time strictly asynchronous, randomized wormholes. The paper presents the first formalization of this type of model with I/O automata, which shows to be an adequate formalism to model hybrid systems.

References

- [1] A. Adelsbach, D. Alessandri, C. Cachin, S. Creese, Y. Deswarte, K. Kursawe, J. C. Laprie, D. Powell, B. Randell, J. Riordan, P. Ryan, W. Simmonds, R. Stroud, P. Verissimo, M. Waidner, and A. Wespi. *Conceptual Model and Architecture of MAFTIA. Project MAFTIA deliverable D21*. January 2002.
- [2] G. Avoine, F. Gartner, R. Guerraoui, and M. Vukolic. Gracefully degrading fair exchange with security modules. In *Proceedings of the Fifth European Dependable Computing Conference*, volume 3463 of *Lecture Notes in Computer Science*, pages 55–71. Springer-Verlag, April 2005.
- [3] R. Baldoni, J. Helary, M. Raynal, and L. Tanguy. Consensus in Byzantine asynchronous systems. *Journal of Discrete Algorithms*, 1(2):185–210, 2003.
- [4] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30, August 1983.
- [5] G. Bracha. An asynchronous $\lfloor (n - 1)/3 \rfloor$ -resilient consensus protocol. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pages 154–162, August 1984.
- [6] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, pages 123–132, July 2000.
- [7] R. Canetti and T. Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 42–51, 1993.

- [8] M. Castro and B. Liskov. A correctness proof for a practical Byzantine-fault-tolerant replication algorithm. Technical Report MIT/LCS/TM-590, MIT Laboratory for Computer Science, June 1999.
- [9] B. Chor and C. Dwork. Randomization in Byzantine agreement. In *Advances in Computing Research 5: Randomness and Computation*, pages 443–497. JAI Press, 1989.
- [10] M. Correia, N. F. Neves, L. C. Lung, and P. Veríssimo. Low complexity Byzantine-resilient consensus. *Distributed Computing*, 17(3):237–249, 2005.
- [11] M. Correia, N. F. Neves, and P. Veríssimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *Computer Journal*, 41(1):82–96, Jan 2006.
- [12] M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Proceedings of the Fourth European Dependable Computing Conference*, pages 234–252, October 2002.
- [13] A. Doudou, B. Garbinato, and R. Guerraoui. Encapsulating failure detection: From crash-stop to Byzantine failures. In *International Conference on Reliable Software Technologies*, pages 24–50, May 2002.
- [14] A. Doudou and A. Schiper. Muteness detectors for consensus with Byzantine processes. Technical Report 97/30, EPFL, 1997.
- [15] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.
- [16] P. Ezhilchelvan, A. Mostefaoui, and M. Raynal. Randomized multivalued consensus. In *Proceedings of the 4th IEEE International Symposium on Object-Oriented Real-Time Computing*, pages 195–200, May 2001.
- [17] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
- [18] J. S. Fraga and D. Powell. A fault- and intrusion-tolerant file system. In *Proceedings of the 3rd International Conference on Computer Security*, pages 203–218, August 1985.
- [19] A. Frier, P. Karlton, and P. Kocher. The SSL 3.0 protocol. Netscape Communications Corp., November 1996.
- [20] M. Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold, 1988.
- [21] R. Guerraoui and A. Schiper. The generic consensus service. *IEEE Transactions on Software Engineering*, 27(1):29–41, January 2001.
- [22] N. Itoi and P. Honeyman. Smartcard integration with Kerberos v5. In *Proceedings of the USENIX Workshop on Smartcard Technology*, May 1999.
- [23] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Mateo, CA, 1996.
- [24] N. Lynch and M. Tuttle. An Introduction to Input/Output automata. *CWI-Quarterly*, 2(3):219–246, Sep 1989.

- [25] D. Malkhi and M. Reiter. Unreliable intrusion detection in distributed computations. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 116–124, June 1997.
- [26] F. Meyer and D. Pradhan. Consensus with dual failure modes. In *Proceedings of the 17th IEEE International Symposium on Fault-Tolerant Computing*, pages 214–222, July 1987.
- [27] H. Moniz, M. Correia, N. F. Neves, and P. Verissimo. Randomized intrusion-tolerant asynchronous services. In *Proceedings of the International Conference on Dependable Systems and Networks*, June 2006. Accepted for publication, to appear.
- [28] A. Mostefaoui, M. Raynal, and F. Tronel. From binary consensus to multivalued consensus in asynchronous message-passing systems. *Information Processing Letters*, (73):207–212, 2000.
- [29] N. F. Neves, M. Correia, and P. Verissimo. Solving vector consensus with a wormhole. *IEEE Transactions on Parallel and Distributed Systems*, 16(12):1120–1131, 2005.
- [30] M. O. Rabin. Randomized Byzantine generals. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 403–409, November 1983.
- [31] V. Shoup and A. D. Rubin. Session key distribution using smart cards. In Springer-Verlag, editor, *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques, Eurocrypt'96*, volume 1070 of *Lecture Notes in Computer Science*, May 1996.
- [32] C. Stueble. Development of a prototype for a security platform for mobile devices. Master's thesis, Universitat des Saarlandes, April 2000.
- [33] J. D. Tygar and B. S. Yee. Dyad: A system for using physically secure coprocessors. In *Proceedings of the Joint Harvard-MIT Workshop on Technological Strategies for the Protection of Intellectual Property in the Network Multimedia Environment*, April 1993.
- [34] P. Verissimo. Uncertainty and predictability: Can they be reconciled? In *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 108–113. Springer-Verlag, 2003.
- [35] P. Verissimo, A. Casimiro, and C. Fetzer. The Timely Computing Base: Timely actions in the presence of uncertain timeliness. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 533–542, June 2000.
- [36] P. Verissimo, N. F. Neves, and M. Correia. Intrusion-tolerant architectures: Concepts and design. In R. Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems*, volume 2677 of *Lecture Notes in Computer Science*, pages 3–36. Springer-Verlag, 2003.

A Correctness Proofs

A.1 Correctness proof for Theorem 1

Proof (sketch): The protocol is based on a consensus protocol executed by the wormholes. This protocol is defined in terms of the properties in Section 2.2. For a typical fail-stop consensus protocol, the properties are satisfied if no more than $\lfloor \frac{n-1}{2} \rfloor$ out of n wormholes fail, an immediate consequence of the assumption than no more than $\lfloor \frac{n-1}{3} \rfloor$ processes fail (Section 2.1).

Validity-2. When a value is proposed, process p_i gives it to the wormhole w_i (Aut.1:1-2,7-9) that sends it to every other wormhole, including itself (Aut.2:1-2,9-11). Then, the wormhole waits for $(n - f)$ messages with values proposed by different wormholes (Aut.2:3-4,12-14). Wormholes are fail-stop so they either send the message once, or do not send it at all. At most f processes can fail, therefore at least $f + 1$ messages come from correct processes: $(f = \lfloor \frac{n-1}{3} \rfloor) \Rightarrow (n - 2f \geq f + 1)$. The property Validity-2 assumes all correct processes propose v , therefore all vectors given to FSC contain at least $f + 1$ copies of v (Aut.2:12-14). The vector decided by FSC, which is one of the vectors proposed, is returned to the process (Aut.2:5-6,15-17), which returns the value that appears at least $f + 1$ times in the vector, i.e., v (Aut.1:3-4,12-14). When the correct processes do not propose the same value, the protocol decides a default value \perp (Aut.1:13-15).

Agreement. The proof derives trivially from the fact that all non-crashed wormholes return the same vector (Aut.2:5-6,15-17), which is used to decide deterministically the value returned (Aut.1:3-4,10-15). Note that the function $\#$ is deterministic even if there are two values v_1 and v_2 such that $\#_{v_1}(Vect) = \#_{v_2}(Vect) \geq f + 1$. In that case, the value among v_1 and v_2 that appears first in $Vect$ is returned.

Termination. An inspection of the two algorithms shows that the protocol terminates if two conditions are satisfied. The first is that at least $n - f$ wormholes have to broadcast the values proposed by the corresponding processes (Aut.2:13). This must happen since at least that number of processes are correct. The second condition is that the FSC consensus has to terminate, something that is guaranteed by the property of Termination of that protocol (Section 2.2). \square

A.2 Correctness proof for Theorem 2

Proof (sketch): The protocol is very similar to the consensus protocol, so the proofs of Agreement and Termination are also the same as in Theorem 1.

Validity-VC. Property 1. If p_i is correct then w_i gets the initial value v (Aut.3:1-2,7-9; Aut.2:1-2),

which it broadcasts to all wormholes in Υ (Aut.2:9-11). Every wormhole proposes to FSC a vector with the default value (\perp) or the broadcasted value v in the entry corresponding to p_i (Aut.2:3-4). FSC simply decides on one of the vectors proposed by the wormholes, and this is the vector decided by the protocol (Aut.2:5-6,15-17; Aut.3:3-4,10-12) therefore the property is satisfied.

Validity-VC. Property 2. We proved that the vector decided is one of the vectors proposed to FSC by one of the wormholes. These vectors include at least $n - f$ entries filled (Aut.2:12-14) and at most f of these entries contain values from failed processes. The property is satisfied since $f = \lfloor \frac{n-1}{3} \rfloor \Rightarrow n - f - f \geq f + 1$. \square

B Fail-stop $\lfloor (n - 1)/3 \rfloor$ -Resilient Binary Consensus Protocol

This section presents a straightforward modification of Bracha’s Byzantine-resilient binary consensus protocol [5] to tolerate $\lfloor (n - 1)/3 \rfloor$ crash faults. This protocol is used in the evaluation of our Byzantine-resilient consensus protocols (see Section 4). The modification is essentially the removal of the ‘reliable broadcast’ primitive and the ‘correctness enforcement’ scheme used in [5] to constrain the behavior of malicious processes. We also generalize Bracha’s assumption of $n = 3f + 1$ to $f = \lfloor (n - 1)/3 \rfloor$.

The protocol is presented in Algorithm 1 following the original format. Also following the original presentation, the algorithm does not terminate when a decision is made. This can be done by making the processes that decided broadcast a halting message.

Theorem 3 *If at most $f = \lfloor \frac{n-1}{3} \rfloor$ processes are stopped, then the protocol presented in Algorithm 1 solves consensus as specified by properties Validity-1, Agreement and Termination.*

Proof (sketch):

Validity-1. The protocol is binary, therefore only two values can be proposed. The property would be false only if all processes proposed 0 (resp. 1) and a correct process decided 1 (resp. 0). A simple inspection of the protocol shows that this is impossible: if all processes propose the same value then all decide it.

Agreement. Two processes cannot decide different values (0 and 1) in the same round since they would need respectively $n - f$ ($d, 0$) messages and $n - f$ ($d, 1$) messages. This is clearly impossible since each process can only send one message per round and step, and $n - f + n - f > n$.

Now, without loss of generality assume process p_0 decides 0 in round k and process p_1 decides 1 in round $k' > k$. Process p_0 must have received $n - f$ ($d, 0$) messages in step 3 of round k so all other

Algorithm 1 Fail-stop $\lfloor (n-1)/3 \rfloor$ -Resilient Binary Consensus Protocol

i_p is set to the value proposed by the process before the first round.

(d, v) is a special value that is used to try to decide v .

Round(k): (by process p)

1. *Broadcast*(i_p) and wait for $n - f$ messages.

$i_p :=$ majority value of the messages.

2. *Broadcast*(i_p) and wait for $n - f$ messages.

If more than $n/2$ of the messages have the same value, then $i_p := (d, v)$.

3. *Broadcast*(i_p) and wait for $n - f$ messages.

If there are at least $(n - f)$ (d, v) messages then *Decide* v .

If there are at least $(n - 2f)$ (d, v) messages then $i_p := v$.

Otherwise, $i_p := 1$ or 0 with probability $1/2$.

Go to step 1 of round $k + 1$.

processes received at most f $(d, 1)$ messages in the same round/step. Therefore, all other processes set their variable i to v since all received at least $n - 2f$ $(d, 0)$ messages (step 3). An inspection of the protocol shows that if all processes set i to v in round k then in round $k + 1$ process p_1 decides 0. A contradiction.

Termination. An inspection shows that the protocol cannot deadlock. We just proved that if a process decides v then all correct processes decide v not after the next round. If no process decides, there is an increasing probability that eventually $n - f$ processes set i to the same value v in step 3 (say, in round k). When this happens, all processes receive at least $n - 2f$ messages with v in step 1 of round $k + 1$ (since only f processes may broadcast a different value) and set i to v . In step 2, all processes broadcast v and set i to (d, v) . Finally, in step 3 all decide. \square

Storage Tradeoffs in a Collaborative Backup Service for Mobile Devices[†]

Ludovic Courtès Marc-Olivier Killijian David Powell

first-name.last-name@laas.fr
LAAS-CNRS
7 avenue du Colonel Roche
31077 Toulouse cedex 4
France

Abstract

Mobile devices are increasingly relied on but are used in contexts that put them at risk of physical damage, loss or theft. We consider a fault-tolerance approach that exploits spontaneous interactions to implement a collaborative backup service. We define the constraints implied by the mobile environment, analyze how they translate into the storage layer of such a backup system and examine various design options. The paper concludes with a presentation of our prototype implementation of the storage layer, an evaluation of the impact of several compression methods, and directions for future work.

1. Introduction

Embedded computers are becoming widely available, in various portable devices such as PDAs, digital cameras, music players and laptops. Most of these devices are now able to communicate using wireless network technologies such as IEEE 802.11, Bluetooth, or Zigbee. Users use such devices to capture more and more data and are becoming increasingly dependent on them. Backing up the data stored on these devices is often done in an *ad hoc* fashion: each protocol and/or application has its own synchronization facilities that can be used when a sister device, usually a desktop computer, is reachable. However, newly created data may be held on the mobile device for a long time before it can be copied. This may be a serious issue since the contexts in which mobile devices are used increase the risks of them being lost, stolen or broken.

Our goal is to leverage the ubiquity of communicating mobile devices to implement a *collaborative* backup service. In such a system, devices participating in the service would be able to use other devices' storage to back up their own data. Of course, each device would have to contribute some of its own storage resources for others to be able to benefit from the service.

Internet-based peer-to-peer systems paved the way for such services. They showed that excess resources available at the peer hosts could be leveraged to support wide-scale resource sharing. Although the amount of resources available on a mobile device is significantly smaller than that of a desktop machine, we believe that this is not a barrier to the creation of mobile peer-to-peer services. They have also shown that wide-scale services could be created without relying on any infrastructure (other than the Internet itself), in a decentralized, self-administered way. From a fault-tolerance viewpoint, peer-to-peer systems provide a high diversity of nodes with independent failure modes [13].

In a mobile context, we believe there are additional reasons to use a collaborative service. For instance, access to a cell phone communication infrastructure (GPRS, UMTS, etc.) may be costly (especially for non-productive data transmission “just” for the sake of backup) while proximity communications are not (using 802.11, Bluetooth, etc.). Similarly, short-distance communication technologies are often more efficient than long-distance ones: they offer a higher throughput and often require less energy. In some scenarios, infrastructure-based networks are simply not available but neighboring devices might be accessible using single-hop communications, or by *ad hoc* routing.

Our target service raises a number of interesting issues, in particular relating to trust management, resource accounting and cooperation incentives. It raises novel issues due to, for instance, mostly-disconnected operation and the consequent difficulty of resorting to centralized

[†]This work was partially supported by the MoSAIC project (ACI S&I, French national program for Security and Informatics; see <http://www.laas.fr/mosaic/>) and the Hidenets project (EU-IST-FP6-26979).

or on-line solutions. A preliminary analysis of these issues may be found in [6,14]. In this paper, the focus is on the mechanisms employed at the storage layer of such a service. We investigate the various design options at this layer and discuss potential trade-offs.

In Section 2, we will detail the requirements of the cooperative backup service on the underlying storage layer. Section 3 presents several design options for this layer based on the current literature and the particular needs that arise from the kind of devices we target. In Section 4, using a prototype of this storage layer, we will evaluate some storage layer algorithms and discuss the necessary tradeoffs. Finally, we will conclude on our current work and sketch future research directions.

2. Collaborative Backup for Mobile Devices

This section gives an overview of the service envisaged and related work. Then we describe the requirements we have identified for the storage layer of the service.

2.1. Design Overview and Related Work

Our goal is to design and implement a collaborative backup system for communicating mobile devices. In this model, mobile devices can play the role of a *contributor*, i.e., a device that offers its storage resources to store data on behalf of other nodes, and a *data owner*, i.e., a mobile device asking a contributor to store some of its data on its behalf. Practically, nodes are expected to contribute as much as they benefit from the system; therefore, they should play both roles at the same time.

For the service to effectively leverage the availability of neighboring communicating devices, the service has to be functional even in the presence of *mutually suspicious device users*. We want users with no prior trust relationships to be able to use the service and to contribute to it harmlessly. This is in contrast with traditional habits where users usually back up their mobile devices' data only on machines they trust, such as their workstation.

This goal also contrasts with previous work on collaborative backup for a personal area network (PAN), such as FlashBack [19], where participating devices are all trustworthy since they belong to the same user. However, censorship-resistant peer-to-peer file sharing systems such as GUNet [2] have a similar approach to security in the presence of adversaries.

Recently, a large amount of research has gone into the design and implementation of Internet-based peer-to-peer backup systems that do not assume prior trust relationships among participants [1,7,9]. There is, however, a significant difference between those Internet-based

systems and what we envision: *connectivity*. Although these Internet-based collaborative backup systems are designed to tolerate disconnections, they do assume a high-level of connectivity. Disconnections are assumed to be mostly transient, whether they be non-malicious (a peer goes off-line for a few days or crashes) or malicious (a peer purposefully disconnects in order to try to benefit from the system without actually contributing to it).

In the context of mobile devices interacting spontaneously, connections are by definition short-lived, unpredictable, and very variable in bandwidth and reliability. Worse than that, a pair of peers may have a chance encounter and start exchanging data, and then never meet again.

To tackle this issue, we assume that each mobile device can at least *intermittently* access the Internet. The backup software running in those mobile devices is expected to take advantage of such an opportunity by re-establishing contacts with (proxies of) mobile devices encountered earlier. For instance, a contributor may wish to send data stored on behalf of another node to some sort of *repository* associated with the owner of the data. Contributors can thus asynchronously *push* data back to their owners. The repository itself can be implemented in various ways: an email mailbox, an FTP server, a fixed peer-to-peer storage system, etc. Likewise, data owners may sometimes need to query their repository as soon as they can access the Internet in order to *pull* back (i.e., restore) their data.

In the remainder of this paper, we will focus on the design of the storage layer of this service on both the data owner and contributor sides.

2.2. Requirements of the Storage Layer

We have identified the following requirements for the mechanisms employed at the storage layer.

Storage efficiency. Backing up data should be as efficient as possible. Data owners should neither ask contributors to store more data than necessary nor send excessive data over the wireless interface. Failing to do so will waste energy and result in inefficient utilization of the storage resources available in the node's vicinity. Inefficient storage may have a strong impact on energy consumption since (i) storage costs translate into transmission costs and (ii) energy consumption on mobile devices is dominated by wireless communication costs, which in turn increase as more data are transferred [28]. *Compression techniques* are thus a key aspect of the storage layer on the data owner side.

Small data blocks. Both the occurrence of encounters of a peer within radio range and the lifetime of the resulting connections are unpredictable. Consequently, the backup application running on a data owner's device

must be able to conveniently split the data to be backed up into small pieces to ensure that it can actually be transferred to contributors. Ideally, data blocks should be able to fit within the underlying network layer's maximum transmission unit or MTU (2304 octets for IEEE 802.11); larger payloads get fragmented into several packets, which introduces overhead at the MAC layer, and possibly at the transport layer too.

Backup atomicity. Unpredictability and the potentially short lifetime of connections, compounded with the possible use of differential compression to save storage resources, mean that it is unlikely to be practical to store a set of files, or even one complete file, on a single peer. Indeed, it may even be undesirable to do so in order to protect data confidentiality [8]. Furthermore, it may be the case that files are modified before their previous version has been completely backed up.

The dissemination of data chunks as well as the coexistence of several versions of a file must not affect backup consistency as perceived by the end-user: a file should be either retrievable *and* correct, or unavailable. Likewise, the distributed store that consists of various contributors shall remain in a "legal" state after new data are backed up onto it. This corresponds to the *atomicity* and *consistency* properties of the ACID properties commonly referred to in transactional database management systems.

Error detection. Accidental modifications of the data are assumed to be handled by the various lower-level software and hardware components involved, such as the communication protocol stack, the storage devices themselves, the operating system's file system implementation, etc. However, given that data owners are to hand their data to untrusted peers, the storage layer must provide mechanisms to ensure that *malicious* modifications to their data are detected with a high probability.

Encryption. Due to the lack of trust in contributors, data owners may wish to encrypt their data to ensure privacy. While there exist scenarios where there is sufficient trust among the participants such that encryption is not compulsory (e.g., several people in the same working group), encryption is a requirement in the general case.

Backup redundancy. Redundancy is the *raison d'être* of any data backup system, but when the system is based on cooperation, the backups themselves must be made redundant. First, the cooperative backup software must account for the fact that contributors may crash accidentally. Second, contributor availability is unpredictable in a mobile environment without continuous Internet access. Third, contributors are not fully trusted and may behave maliciously. Indeed, the literature on Internet-based peer-to-peer backup systems describes a range of

attacks against data availability, ranging from data retention (i.e., a contributor purposefully refuses to allow a data owner to retrieve its data) to selfishness (i.e., a participant refuses to spend energy and storage resources storing data on behalf of other nodes) [7,9]. All these uncertainties make redundancy even more critical in a cooperative backup service for mobile devices.

3. Design Options for the Storage Layer

In this section, we present design options able to satisfy each of the requirements identified for above.

3.1. Storage Efficiency

In wired distributed cooperative services, storage efficiency is often addressed by ensuring that a given content is only stored once. This property is known as *single-instance storage* [4]. It can be thought of as a form of compression among several data units. In a file system, where the "data unit" is the file, this means that a given content stored under different file names will be stored only once. On Unix-like systems, revision control and backup tools implement this property by using hard links [20,25]. It may also be provided at a sub-file granularity, instead of at a whole file level, allowing reduction of unnecessary duplication with a finer-grain.

Archival systems [23,35], peer-to-peer file sharing systems [2], peer-to-peer backup systems [7], network file systems [22], and remote synchronization tools [31] have been demonstrated to benefit from single-instance storage, either by improving storage efficiency or reducing bandwidth.

Compression based on resemblance detection, i.e., *differential compression*, or *delta encoding*, is unsuitable for our environment since (i) it requires access to all the files already stored, (ii) it is CPU- and memory-intensive, and (iii) the resulting *delta chains* weaken data availability [15,35].

Traditional lossless compression (i.e., *zip* variants), allows the elimination of duplication *within* single files. As such, it naturally complements inter-file and inter-version compression techniques [35]. Section 4 contains a discussion of the combination of both techniques in the framework of our proposed backup service. Lossless compressors usually yield better compression when operating on large input streams [15] so compressing concatenated files rather than individual files improves storage efficiency [35]. However, we did not consider this approach suitable for mobile device backup since it may be more efficient to backup only those files (or part of files) that have changed.

There exist a number of application-specific compression algorithms, such as the *lossless* algorithms used

by the FLAC audio codec, the PNG image format, and the XMill XML compressor [17]. There is also a plethora of *lossy* compression algorithms for audio samples, images, videos, etc. While using such application-specific algorithms might be beneficial in some cases, we have focused instead on generic lossless compression.

3.2. Small Data Blocks

We now consider the options available to: (1) chop input streams into small blocks, and (2) create appropriate meta-data describing how those data blocks should be reassembled to produce the original stream.

3.2.1. Chopping Algorithms

As stated in Section 2.2, the size of blocks that are to be sent to contributors of the backup service has to be bounded, and preferably small, to match the nature of peer interactions in a mobile environment. There are several ways to cut input streams into blocks. Which algorithm is chosen has an impact on the improvement yielded by single-instance storage applied at the block level.

Splitting input streams into fixed-size blocks is a natural solution. When used in conjunction with a single-instance storage mechanism, it has been shown to improve the compression across files or across file versions [23]. Manber proposed an alternative content-based stream chopping algorithm [21] that yields better duplication detection across files, a technique sometimes referred to as *content-defined blocks* [15]. The algorithm determines block boundaries by computing Rabin fingerprints on a sliding window of the input streams. Thus, it only allows the specification of an *average* block size (assuming random input). Various applications such as archival systems [35], network file systems [22] and backup systems [7] benefit from this algorithm. Section 4 provides a comparison of both algorithms.

3.2.2. Stream Meta-Data

Placement of stream meta-data. Stream meta-data is information that describes which blocks comprise the stream and how they should be reassembled to produce the original stream. Such meta-data can either be embedded along with each data block or stored separately. The main evaluation criteria of a meta-data structure are read efficiency (e.g., algorithmic complexity of stream retrieval, number of accesses needed) and size (e.g., how the amount of meta-data grows compared to data).

We suggest a more flexible approach whereby stream meta-data (i.e., which blocks comprise a stream) is separated both from file meta-data (i.e., file name, permissions, etc.) and the file content. This has several advantages. First, it allows a data block to be referenced

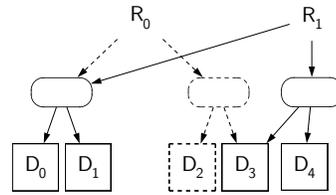


Figure 1. A tree structure for stream meta-data. Leaves represent data blocks while higher blocks are meta-data blocks.

multiple times and hence allows for single-instance storage at the block level. Second, it promotes *separation of concerns*. For instance, file-level meta-data (e.g., file path, modification time, permissions) may change without having to modify the underlying data blocks, which is important in scenarios where propagating such updates would be next to impossible. Separating meta-data and data also leaves the possibility of applying the same “filters” (e.g., compression, encryption), or to use similar redundancy techniques for both data and meta-data blocks. This will be illustrated in Section 4. This approach is different from the one used in Hydra [34] but not unlike that of OpenCM [27].

Indexing individual blocks. The separation of data and meta-data means that there must be a way for meta-data blocks to refer to data blocks: data blocks must be indexed or *named*¹. The block naming scheme must fulfill several requirements. First, it must not be based on non-backed-up user state which would be lost during a crash. Most importantly, the block naming scheme must guarantee that *name clashes* among the blocks of a data owner cannot occur. In particular, block IDs must remain valid in time so that a given block ID is not wrongfully re-used when a device restarts the backup software after a crash. Given that data blocks will be disseminated among several peers and will ultimately migrate to their owner’s repository, blocks IDs should remain valid in space, that is, they should be independent of contributor names. This property also allows for *pre-computation* of block IDs and meta-data blocks: stream chopping and indexing do not need to be done upon a contributor encounter, but can be performed *a priori*, once for all. This saves CPU time and energy, and allows data owners to immediately take advantage of a backup opportunity. A practical naming scheme widely used in the literature will be discussed in Section 3.4.

Indexing sequences of blocks. Byte streams (file contents) can be thought of as sequences of blocks.

¹In the sequel we use the terms “block ID”, “name”, and “key” interchangeably.

Meta-data describing the list of blocks comprising a byte stream need to be produced and stored. In their simplest form, such meta-data are a vector of block IDs, or in other words, a byte stream. This means that this byte stream can in turn be indexed, recursively, until a meta-data byte stream is produced that fits the block size constraints. This approach yields the meta-data structure shown in Figure 1 which is comparable to that used by Venti and GUNet [2,23].

Contributor interface. With such a design, contributors do not need to know about the actual implementation of block and stream indexing used by their clients, nor do they need to be aware of the data/meta-data distinction. All they need to do is to provide primitives of a keyed block storage:

- `put (key, data)` inserts the data block `data` and associates it with `key`, a block ID chosen by the data owner according to some naming scheme;
- `get (key)` returns the data associated with `key`.

This simple interface suffices to implement, on the data owner side, byte stream indexing and retrieval. Also, it is suitable for an environment in which service providers and users are mutually suspicious because it places as little burden as possible on the contributor side. The same approach was adopted by Venti [23] and by many peer-to-peer systems [2,7].

3.3. Backup Atomicity

Distributed and mobile file systems such as Coda [16] which support concurrent read-write access to the data and do not have built-in support for revision control, differ significantly from backup systems. Namely, they are concerned about update propagation and reconciliation in the presence of concurrent updates. Not surprisingly, a read-write approach does not adapt well to the loosely connected scenarios we are targeting: data owners are not guaranteed to meet *every* contributor storing data on their behalf in a timely fashion, which makes update propagation almost impossible. Additionally, it does not offer the desired atomicity requirement discussed in Section 2.2.

The *write once or append only* semantics adopted by archival [11,23], backup [7,25] and versioning systems [20,26,27] solve these problems. Data is always appended to the storage system, and never modified in place. This is achieved by assigning each piece of data a unique identifier. Therefore, insertion of content (i.e., data blocks) into the storage mechanism (be it a peer machine, a local file system or data repository) is atomic. Because data is only added, never modified, consistency is also guaranteed: insertion of a block cannot result in an inconsistent state of the storage mechanism.

A potential concern with this approach is its cost in terms of storage resources. It has been argued, however, that the cost of storing subsequent revisions of whole sets of files can be very low provided inter-version compression techniques like those described earlier are used [10,23,26]. In our case, once a contributor has finally transferred data to their owner's repository, it may reclaim the corresponding storage resources, which further limits the cost of this approach.

From an end-user viewpoint, being able to restore an old copy of a file is more valuable than being unable to restore the file at all. All these reasons make the write-only approach suitable to the storage layer of our cooperative backup service.

3.4. Error Detection

Error-detecting codes can be computed either at the level of whole input streams or at the level of data blocks. They must then be part of, respectively, the stream meta-data, or the block meta-data. We argue the case for cryptographic hash functions as a means of providing the required error detection and as a block-level indexing scheme.

Cryptographic hash functions. The error-detecting code we use must be able to detect *malicious* modifications. This makes error-detecting codes designed to tolerate random, accidental faults inappropriate. We must instead use *collision-resistant* and *preimage-resistant* hash functions, which are explicitly designed to detect tampering [5].

Along with integrity, *authenticity* of the data must also be guaranteed, otherwise a malicious contributor could deceive a data owner by producing fake data blocks along with valid cryptographic hashes. Thus, digital signatures should be used to guarantee the authenticity of the data blocks. Fortunately, not all blocks need to be signed: signing a root meta-data block (as shown in Figure 1) is sufficient. This is similar to the approach taken by OpenCM [27]. Note, however, that while producing random data blocks and their hashes is easy, producing the corresponding meta-data blocks is next to impossible without knowing what particular meta-data schema is used by the data owner.

Content-based indexing. Collision-resistant hash functions have been assumed to meet the requirements of a data block naming scheme as defined in Section 3.2.2, and to be a tool allowing for efficient implementations of single-instance storage [7,22,23,29,31,35]. In practice, these implementations assume that whenever two data blocks yield the same cryptographic hash value, their contents *are* identical. Given this assumption, implementation of a single-instance store is straightforward: a block only needs to be stored if its hash val-

ue was not found in the locally maintained block hash table.

In [12], Henson argues that accidental collisions, although extremely rare, do have a slight negative impact on software reliability and yield silent errors. Given an n -bit hash output produced by one of the functions listed above, the expected workload to generate a collision out of two *random* inputs is of the order of $2^{n/2}$ [5]. More precisely, if we are to store, say, 8 GiB of data in the form of 1 KiB blocks, we end up with 2^{43} blocks, whereas SHA-1, for instance, would require 2^{80} blocks to be generated on average before an accidental collision occurs. We consider this to be reasonable in our application since it does not impede the tolerance of faults in any significant way. Also, Henson's fear of *malicious* collisions does not hold given the preimage-resistance property provided by the commonly-used hash functions².

Content-addressable storage (CAS) thus seems a viable option for our software layer as it fulfills both the error-detection and data block naming requirements. In [29], the authors assume a block ID space shared across all CAS users and providers. In our scenario, CAS providers (contributors) do not trust their clients (data owners) so they need either to enforce the block naming scheme (i.e., make sure that the ID of each block is indeed the hash value of its content), or to maintain a per-user name space.

3.5. Encryption

Data encryption may be performed either at the level of individual blocks, or at the level of input streams. Encrypting the input stream *before* it is split into smaller blocks breaks the single-instance storage property at the level of individual blocks. This is because encryption aims to ensure that the encrypted output of two similar input streams will not be correlated.

Leaving input streams unencrypted and encrypting individual blocks yielded by the chopping algorithm does not have this disadvantage. More precisely, it preserves single-instance storage at the level of blocks at least *locally*, i.e., on the client side. If asymmetric ciphering algorithms are used, the single-instance storage property is no longer ensured *across* peers, since each peer encrypts data with its own private key. However, we do not consider this a major drawback for the majority of scenarios considered where little or no data are common to several participants. Moreover, solutions to this problem exist, notably *convergent encryption* [7].

²The recent attacks found on SHA-1 by Wang et al. [33] do not affect the preimage-resistance of this function.

3.6. Backup Redundancy

Replication strategies. Redundancy management in the context of our collaborative backup service for mobile devices introduces a number of new challenges. Peer-to-peer file sharing systems are not a good source of inspiration in this respect given that they rely on redundancy primarily as a means of reducing access time to popular content [24].

For the purposes of fault-tolerance, statically-defined redundancy strategies have been used in Internet-based scenarios where the set of servers responsible for holding replicas is known *a priori*, and where servers are usually assumed to be reachable “most of the time” [8,34]. Internet-based peer-to-peer backup systems [7,9] have relaxed these assumptions. However, although they take into account the fact that contributors may become unreachable, strong connectivity assumptions are still made: the inability to reach a contributor is assumed to be the exception, rather than the rule. As a consequence, unavailability of a contributor is quickly interpreted as a symptom of malicious behavior [7,9].

The connectivity assumption does not hold in our case. Additionally, unlike with Internet-based systems, the very encounter of a contributor is unpredictable. This has a strong impact on the possible replication strategies, and on the techniques used to implement redundancy.

Erasure codes have been used as a means to tolerate failures of storage sites while being more storage-efficient than simple replication [34]. Usually, (n,k) erasure codes are defined as follows [18,34]:

- an (n,k) code maps a k -symbol block to an n -symbol codeword;
- $k + \epsilon$ symbols suffice to recover the exact original data; the code is *optimal* when $\epsilon = 0$;
- optimal (n,k) schemes tolerate the loss of $(n - k)$ symbols and have an effective storage use of k/n .

Such an approach seems very attractive to improve storage efficiency while still maximizing data availability.

However, as argued in [3,18,32], an (n,k) scheme with $k > 1$ can hinder data availability because it requires k peers to be available for data to be retrieved, instead of just 1 with mirroring (i.e., an $(n,1)$ scheme). Also, given the unpredictability of contributor encounters, a scheme with $k > 1$ is risky since a data owner may fail to store k symbols on different contributors. On the other hand, from a confidentiality viewpoint, increasing dissemination and purposefully placing less than k symbols on any given untrusted contributor may be a good strategy [8]. Intermediate solutions can also be imagined, e.g., mirroring blocks that have never been replicated and choosing

$k > 1$ for blocks already mirrored at least once. This use of different *levels of dispersal* was also mentioned by the authors of InterMemory [11] as a way to accommodate contradictory requirements. Finally, a dynamically adaptive behavior of erasure coding may be considered as [3] suggests.

Replica scheduling and dissemination. As stated in Section 2.2, it is plausible that a file will be only partly backed up when a newer version of this file enters the backup creation pipeline. One could argue that the replica scheduler should finish distributing the data blocks from the old version before distributing those of the new version. This policy would guarantee, at least, availability of the old version of the file. On the other hand, in certain scenarios, users might want to favor freshness over availability, i.e., they might request that newer blocks are scheduled first for replication.

This clearly illustrates that a wide range of *replica scheduling and dissemination policies and corresponding algorithms* can be defended depending on the scenario considered. At the core of a given replica scheduling and dissemination algorithm is a *dispersal function* that decides on a level of dispersal for any given data block. The algorithm must evolve *dynamically* to account for several changing factors. In FlashBack [19], the authors identify a number of important factors that they use to define a *device utility function*. Those factors include *locality* (i.e., the likelihood of encountering a given device again later) as well as *power and storage resources* of the device.

In addition to those factors, our backup software needs to account for the current level of trust in the contributor at hand. If a data owner fully trusts a contributor, e.g., because it has proven to be well-behaved over a given period of time, the data owner may choose to store complete replicas (i.e., mirrors) on this contributor.

4. Preliminary Evaluation

This section presents our prototype implementation of the storage layer of the envisaged backup system, as well as a preliminary evaluation of key aspects.

4.1. Implementation Overview

We have implemented a prototype of the storage layer discussed above, a basic building block of the cooperative backup framework we are designing. This layer is performance-critical and we implemented it in C. The resulting library, `libchop`, consists of 7 K physical source lines of code. It was designed to be flexible enough so that different techniques could be combined and evaluated, by providing a few well-defined interfaces as shown in Figure 2. The library itself is not concerned

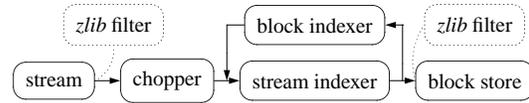


Figure 2. Data flow in the `libchop` backup creation pipeline.

with the backup of file system-related meta-data such as file paths, permissions, etc. Implementing this is left to higher-level layers akin to OpenCM’s schemas [27].

Implementations of the `chopper` interface chop input streams into small fixed-size blocks, or according to Manber’s algorithm [21]. Block indexers name blocks and store them in a keyed block store (e.g., an on-disk database). The `stream_indexer` interface provides a method that iterates over the blocks yielded by the given `chopper`, indexes them, produces corresponding meta-data blocks, and stores them in a block store. In the proposed cooperative backup service, chopping and indexing are to be performed on the data owner side, while the block store itself will be realized by contributors. Finally, `libchop` also provides *filters*, such as `zlib` compression and decompression filters, which may be conveniently reused in different places, for instance between a file-based input stream and a `chopper`, or between a `stream_indexer` and a block store.

In the following experiments, the only `stream_indexer` used is a “tree indexer” as shown in Figure 1. We used an on-disk block store that uses TDB as the underlying database [30]. For each file set, we started with a new, empty database.

4.2. Evaluation of Compression Techniques

Our implementation has allowed us to evaluate more precisely some of the tradeoffs outlined in Section 3. After describing the methodology and workloads that were used, we will comment the results obtained.

4.2.1. Methodology and Workloads

Methodology. The purpose of our evaluation is to compare the various compression techniques described earlier in order to better understand the tradeoffs that must be made. We measured the storage efficiency and computational cost of each method, both of which are critical criteria for resource-constrained devices. The measures were performed on a 500 MHz G4 Macintosh running GNU/Linux (running them on, say, an ARM-based mobile device would have resulted in lower throughputs; however, since we are interested in *comparing* throughputs, this would not make any significant difference).

Name	Size	Files	Avg. Size
Lout (versions 3.20 to 3.29)	76 MiB	5853	13 KiB
Ogg Vorbis files	69 MiB	17	4 MiB
mbox-formatted mailbox	7 MiB	1	7 MiB

Figure 3. File sets.

We chose several workloads and compared the results obtained using different configurations. These file sets, shown in Figure 3, qualify as *semi-synthetic* workloads because they are actual workloads, although they were not taken from a real mobile device. The motivation for this choice was to purposefully target specific file *classes*. The idea is that the results should remain valid for any file of these classes.

File sets. In Figure 3, the first file set contains 10 successive versions of the source code of the Lout document formatting system, i.e., low-density, textual input (C and Lout code), spread across a number of small files. Of course, this type of data is not typical of mobile devices like PDAs and cell phones. Nevertheless, the results obtained with this workload should be similar to those obtained with widely-used textual data format such as XML. The second file set shown in Figure 3 consists of 17 Ogg Vorbis files, a high-density binary format (Ogg Vorbis is a lossy audio compression format), typical of the kind of data that may be found on devices equipped with sampling peripherals. The third file set consists of a single, large file: a mailbox in the Unix mbox format which is an append-only textual format. Such data are likely to be found in a similar form on communicating devices.

Configurations. Figure 4 shows the storage configurations we have used in our experiments. For each configuration, it indicates whether single-instance storage was provided, which chopping algorithm was used and what the expected block size was, as well as whether the input stream or output blocks were compressed using a lossless stream compression algorithm (*zlib* in our case). Our intent is not to evaluate the outcome of each algorithm independently, but rather that of whole configurations. Thus, instead of experimenting with every possible combination, we chose to retain only those that (i) made sense from an algorithmic viewpoint and (ii) were helpful in understanding the tradeoffs at hand.

Configurations A_1 and A_2 serve as baselines for the overall compression ratio and computational cost. Comparing them is also helpful in determining the computational cost due to single-instance storage alone. Subsequent configurations all chop input streams into small blocks whose size fits our requirements (1 KiB, which should yield packets slightly smaller than IEEE 802.11's

Config.	Single Instance?	Chopping Algo.	Expected Block Size	Input Zipped?	Blocks Zipped?
A_1	no	—	—	yes	—
A_2	yes	—	—	yes	—
B_1	yes	Manber's	1024 B	no	no
B_2	yes	Manber's	1024 B	no	yes
B_3	yes	fixed-size	1024 B	no	yes
C	yes	fixed-size	1024 B	yes	no

Figure 4. Description of the configurations experimented.

MTU); they all implement single-instance storage of the blocks produced.

Common octet sequences are unlikely to be found *within* a *zlib*-compressed stream, by definition. Hence, zipping the input precludes advantages to be gained by block-level single-instance storage afterwards. Thus, we did not include a configuration where a zipped input stream would then be passed to a chopper implementing Manber's algorithm.

The B configurations favor sub-file single-instance storage by not compressing the input before chopping it. B_2 improves over B_1 by adding the benefits of *zlib* compression at the block-level. Conversely, configuration C favors traditional lossless compression over sub-file single-instance storage since it applies lossless compression to the input stream.

Our implementation of Manber's algorithm uses a sliding window of 48 B which was reported to provide good results [22]. All configurations but A_1 use single-instance storage, realized using the `libchop` "hash" block indexer that uses SHA-1 hashes as unique block identifiers. For A_1 , a block indexer that systematically provides unique IDs (per RFC 4122) was chosen.

The chosen configurations and file sets are quite similar to those described in [15,35], except that, as explained in Section 3.1, we do not evaluate the storage efficiency of the delta encoding technique proposed therein.

4.2.2. Results

Figure 5 shows the compression ratios obtained for each configuration and each file set. The bars show the ratio of the size of the resulting blocks, *including* meta-data (sequences of SHA-1 hashes), to the size of the input data, for each configuration and each data set. The lines represent the corresponding throughputs.

Impact of the data type. Not surprisingly, the set of Vorbis files defeats all the compression techniques. Most configurations incur a slight storage overhead due to the amount of meta-data generated.

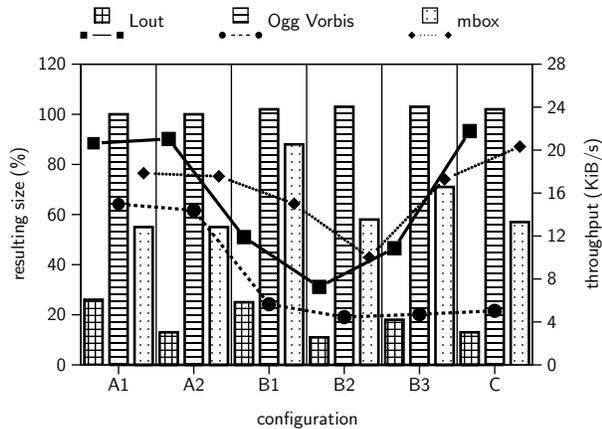


Figure 5. Storage efficiency and computational cost of several configurations.

Impact of single-instance storage. Comparing the results obtained for A_1 and A_2 shows benefits only in the case of the successive source code distributions, where it halves the amount of data stored (13 % vs. 26 %). This is due to the fact that successive versions of the software have a lot of files in common. Furthermore, it shows that single-instance storage implemented using cryptographic hashes does not degrade throughput, which is the reason why we chose to use it in all configurations.

As expected, single-instance storage applied at the block-level is mainly beneficial for the Lout file set where it achieves noticeable inter-version compression, comparable with that produced with *zlib* in A_1 . The best compression ratio overall is obtained with B_2 where individual blocks are *zlib*-compressed. However, the compression ratios obtained with B_2 are comparable to those obtained with C , and only slightly better in the Lout case (11 % vs. 13 %). Thus, we conclude that there is little storage efficiency improvement to be gained from the combination of single-instance storage and Manber’s chopping algorithm compared to traditional lossless compression, especially when applied to the input stream.

The results in [35] are slightly more optimistic regarding the storage efficiency of a configuration similar to B_2 , which may be due to the use a smaller block (512 B) and a larger file set.

Computational cost. Comparing the computational costs of the B configurations with that of C provides an important indication as to which kind of configuration suits our needs best. Indeed, input zipping and fixed-size chopping in C yield a processing throughput three times higher than that of B_2 (except for the set of Vorbis files). Thus, C is the configuration that offers the best tradeoff between computational cost and storage efficiency for low-entropy data.

Additional conclusions can be drawn with respect to throughput. First, the cost of *zlib*-based compression appears to be reasonable, particularly when performed on the input stream rather than on individual blocks, as evidenced, e.g., by B_3 and C . Second, the input data type has a noticeable impact on the computational cost. In particular, applying lossless compression is more costly for the Vorbis files than for low-entropy data. Therefore, it would be worthwhile to disable *zlib* compression for compressed data types.

5. Conclusion and Future Work

In this paper, we have considered the viability of collaboration between peer mobile devices to implement a cooperative backup service. We have identified six essential requirements for the storage layer of such a service, namely: (i) storage efficiency; (ii) small data blocks; (iii) backup atomicity; (iv) error detection; (v) encryption; (vi) backup redundancy. The various design options for meeting these requirements have been reviewed and a preliminary evaluation carried out using a prototype implementation of the storage layer.

Our evaluation has allowed us to assess different storage techniques, both in terms of storage efficiency and computational cost. We conclude that the most suitable combination for our purposes combines the use of lossless input compression with fixed-size chopping and single-instance storage. Other techniques were rejected for providing little storage efficiency improvement compared to their CPU cost.

Future work on the optimization of the storage layer concerns several aspects. First, the energy costs of the various design options need to be assessed, especially those related to the wireless transmission of backup data between nodes. Second, the performance and dependability impacts of various replica scheduling and dissemination strategies need to be evaluated as a function, for example, of the expected frequencies of data updates, cooperative backup opportunities and infrastructure connections. Third, it seems likely that no single configuration of the backup service will be appropriate for all situations, so dynamic adaptation of the service to suit different contexts needs to be investigated.

Finally, the issues relating to trust management, resource accounting and cooperation incentives need to be addressed, especially inasmuch as the envisaged mode of mostly-disconnected operation imposes additional constraints. Current research in this direction, in collaboration with our partners in the MoSAIC project, is directed at evaluating mechanisms such as microeconomic and reputation-based incentives.

References

- [1] C. BATTEN, K. BARR, A. SARAF, S. TREPTIN. pStore: A secure peer-to-peer backup system. MIT-LCS-TM-632, MIT Laboratory for Computer Science, December 2001.
- [2] K. BENNETT, C. GROTHOFF, T. HOROZOV, I. PATRASCU. Efficient Sharing of Encrypted Data. *Proc. of the 7th Australasian Conf. on Information Security and Privacy (ACISP 2002)*, (2384)pp. 107–120, 2002.
- [3] R. BHAGWAN, K. TATI, Y.-C. CHENG, S. SAVAGE, G. M. VOELKER. Total Recall: System Support for Automated Availability Management. *Proc. of the ACM/USENIX NSDI*, 2004.
- [4] W. J. BOLOSKY, J. R. DOUCEUR, D. ELY, M. THEIMER. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. *Proc. of the Int. Conf. on Measurement and Modeling of Computer Systems*, pp. 34–43, 2000.
- [5] NESSIE CONSORTIUM. NESSIE Security Report. NES/DOC/ENS/WP5/D20/2, February 2003.
- [6] L. COURTÈS, M.-O. KILLIJIAN, D. POWELL, M. ROY. Sauvegarde coopérative entre pairs pour dispositifs mobiles. *Actes des deuxièmes journées francophones Mobilité et Ubiquité (UbiMob)*, pp. 97–104, 2005.
- [7] L. P. COX, B. D. NOBLE. Pastiche: Making Backup Cheap and Easy. *5th USENIX OSDI*, pp. 285–298, 2002.
- [8] Y. DESWARTE, L. BLAIN, J.-C. FABRE. Intrusion Tolerance in Distributed Computing Systems. *Proc. of the IEEE Symp. on Research in Security and Privacy*, pp. 110–121, 1991.
- [9] S. ELNIKETY, M. LILLIBRIDGE, M. BURROWS. Peer-to-peer Cooperative Backup System. *The USENIX FAST*, 2002.
- [10] T. J. GIBSON, E. L. MILLER. Long-Term File Activity Patterns in a UNIX Workstation Environment. *Proc. of the 15th IEEE Symp. on MSS*, pp. 355–372, 1998.
- [11] A. V. GOLDBERG, P. N. YIANILOS. Towards an Archival Intermemory. *Proc. IEEE Int. Forum on Research and Technology Advances in Digital Libraries (ADL'98)*, pp. 147–156, 1998.
- [12] V. HENSON. An Analysis of Compare-by-hash. *Proc. of HotOS IX: The 9th HotOS*, pp. 13–18, 2003.
- [13] F. JUNQUEIRA, R. BHAGWAN, K. MARZULLO, S. SAVAGE, G. M. VOELKER. The Phoenix Recovery System: Rebuilding from the Ashes of an Internet Catastrophe. *9th HotOS*, 2003.
- [14] M.-O. KILLIJIAN, D. POWELL, M. BANÂTRE, P. COUDERC, Y. ROUDIER. Collaborative Backup for Dependable Mobile Applications. *Proc. of 2nd Int. Workshop on Middleware for Pervasive and Ad-Hoc Computing (Middleware 2004)*, pp. 146–149, 2004.
- [15] P. KULKARNI, F. DOUGLIS, J. LAVOIE, J. M. TRACEY. Redundancy Elimination Within Large Collections of Files. *Proc. of the USENIX Annual Technical Conf.*, 2004.
- [16] Y.-W. LEE, K.-S. LEUNG, M. SATYANARAYANAN. Operation-based Update Propagation in a Mobile File System. *Proc. of the USENIX Annual Technical Conf.*, pp. 43–56, 1999.
- [17] H. LIEFKE, D. SUCIU. XMill: an Efficient Compressor for XML Data. *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pp. 153–164, 2000.
- [18] W. K. LIN, D. M. CHIU, Y. B. LEE. Erasure Code Replication Revisited. *Proc. of the 4th P2P*, pp. 90–97, 2004.
- [19] B. T. LOO, A. LAMARCA, G. BORRIELLO. Peer-To-Peer Backup for Personal Area Networks. IRS-TR-02-015, UC Berkeley; Intel Seattle Research (USA), May 2003.
- [20] T. LORD. The GNU Arch Distributed Revision Control System. 2005, <http://www.gnu.org/software/gnu-arch/>.
- [21] U. MANBER. Finding Similar Files in a Large File System. *Proc. of the USENIX Winter 1994 Conf.*, pp. 1–10, 1994.
- [22] A. MUTHITACHAROEN, B. CHEN, D. MAZIÈRES. A Low-Bandwidth Network File System. *Proc. of the 18th ACM SOSP*, pp. 174–187, 2001.
- [23] S. QUINLAN, S. DORWARD. Venti: A New Approach to Archival Storage. *Proc. of the 1st USENIX FAST*, pp. 89–101, 2002.
- [24] K. RANGANATHAN, A. IAMNITCHI, I. FOSTER. Improving Data Availability Through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities. *Proc. of the Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, pp. 376–381, 2002.
- [25] M. RUBEL. Rsnapshot: A Remote Filesystem Snapshot Utility Based on Rsync. 2005, <http://rsnapshot.org/>.
- [26] D. S. SANTRY, M. J. FEELEY, N. C. HUTCHINSON, A. C. VEITCH, R. W. CARTON, J. OFIR. Deciding when to forget in the Elephant file system. *Proc. of the 17th ACM SOSP*, pp. 110–123, 1999.
- [27] J. S. SHAPIRO, J. VANDERBURGH. CPCMS: A Configuration Management System Based on Cryptographic Names. *Proc. of the USENIX Annual Technical Conf., FREENIX Track*, pp. 207–220, 2002.
- [28] M. STEMM, P. GAUTHIER, D. HARADA, R. H. KATZ. Reducing Power Consumption of Network Interfaces in Hand-Held Devices. *IEEE Transactions on Communications*, E80-B(8), August 1997, pp. 1125–1131.
- [29] N. TOLIA, M. KOZUCH, M. SATYANARAYANAN, B. KARP, T. BRESSOUD, A. PERRIG. Opportunistic Use of Content Addressable Storage for Distributed File Systems. *Proc. of the USENIX Annual Technical Conf.*, pp. 127–140, 2003.
- [30] A. TRIDGELL, P. RUSSEL, J. ALLISON. The Trivial Database. 1999, <http://samba.org/>.
- [31] A. TRIDGELL, P. MACKERRAS. The Rsync Algorithm. TR-CS-96-05, Department of Computer Science, Australian National University Canberra, Australia, 1996.
- [32] A. VERNOIS, G. UTARD. Data Durability in Peer to Peer Storage Systems. *Proc. of the 4th Workshop on Global and Peer to Peer Computing*, pp. 90–97, 2004.
- [33] X. WANG, Y. YIN, H. YU. Finding Collisions in the Full SHA-1. *Proc. of the CRYPTO Conf.*, pp. 17–36, 2005.
- [34] L. XU. Hydra: A Platform for Survivable and Secure Data Storage Systems. *Proc. of the ACM Workshop on Storage Security and Survivability*, pp. 108–114, 2005.
- [35] L. L. YOU, K. T. POLLACK, AND D. D. E. LONG. Deep Store: An Archival Storage System Architecture. *Proc. of the 21st ICDE*, pp. 804–815, 2005.

Exploring Gafni's reduction land: from Ω^k to wait-free adaptive $(2p - \lceil \frac{p}{k} \rceil)$ -renaming via k -set agreement

Achour MOSTEFAOUI Michel RAYNAL Corentin TRAVERS
IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
{achour | raynal | ctravers}@irisa.fr

May 29, 2006

Abstract

The adaptive renaming problem consists in designing an algorithm that allows p processes (in a set of n processes) to obtain new names despite asynchrony and process crashes, in such a way that the size of the new renaming space M be as small as possible. It has been shown that $M = 2p - 1$ is a lower bound for that problem in asynchronous atomic read/write register systems.

This paper is an attempt to circumvent that lower bound. To that end, considering first that the system is provided with a k -set object, the paper presents a surprisingly simple adaptive M -renaming wait-free algorithm where $M = 2p - \lceil \frac{p}{k} \rceil$. To attain this goal, the paper visits what we call Gafni's reduction land, namely, a set of reductions from one object to another object as advocated and investigated by Gafni. Then, the paper shows how a k -set object can be implemented from a leader oracle (failure detector) of a class denoted Ω^k . To our knowledge, this is the first time that the failure detector approach is investigated to circumvent the $M = 2p - 1$ lower bound associated with the adaptive renaming problem. In that sense, the paper establishes a connection between renaming and failure detectors.

Keywords: Adaptive algorithm, Asynchronous system, Atomic register, Consensus, Divide and conquer, Leader oracle, Renaming, Set agreement, Shared object, Wait-free algorithm.

1 Introduction

The renaming problem The *renaming* problem is a coordination problem initially introduced in the context of asynchronous message-passing systems prone to process crashes [3]. Informally, it consists in the following. Each of the n processes that define the system has an initial name taken from a very large domain $[1..N]$ (usually, $n \ll N$). Initially, a process knows only its name, the value n , and the fact that no two processes have the same initial name. The processes have to cooperate to choose new names from a name space $[1..M]$ such that $M \ll N$ and no two processes obtain the same new name. The problem is then called *M-renaming*.

Let t denote the upper bound on the number of processes that can crash. It has been shown that $t < n/2$ is a necessary and sufficient requirement for solving the renaming problem in an asynchronous message-passing system [3]. That paper presents also a message-passing algorithm whose size of the renaming space is $M = n + t$.

The problem has then received a lot of attention in the context of asynchronous shared memory systems made up of atomic read/write registers. Numerous wait-free renaming algorithms have been designed (e.g., [2, 4, 5, 7, 9, 21]). *Wait-free* means here that a process that does not crash has to obtain a new name in a finite number of its own computation steps, regardless of the behavior of the other processes (they can be

arbitrarily slow or even crash) [18]. Consequently, wait-free implies $t = n - 1$. An important result in such a context, concerns the lower bound on the new name space. It has been shown in [19] that there is no wait-free renaming algorithm when $M < 2n - 1$. As wait-free $(2n - 1)$ -renaming algorithms have been designed, it follows that that $M = 2n - 1$ is a tight lower bound.

The previous discussion implicitly assumes the “worst case” scenario: all the processes participate in the renaming, and some of them crash during the algorithm execution. The net effect of crashes and asynchrony create “noise” that prevents the renaming space to be smaller than $2n - 1$. But it is not always the case that all the processes want to obtain a new name. (A simple example is when some processes crash before requiring a new name.) So, let p , $1 \leq p \leq n$, be the number of processes that actually participate in the renaming. A renaming algorithm guarantees *adaptive* name space if the size of the new name space is a function of p and not of n . Several adaptive wait-free algorithms have been proposed that are optimal as they provide $M = 2p - 1$ (e.g., [2, 4, 9]).

The question addressed in the paper Let us assume that we have a solution to the consensus problem. In that case, it easy to design an adaptive renaming algorithm where $M = p$ (the number of participating processes). The solution is as follows. From consensus objects, the processes build a concurrent queue that provides them with two operations: a classical enqueue operation and a read operation that provides its caller with the current content of the queue (without modifying the queue). Such a queue object has a sequential specification and each operation can always be executed (they are *total* operations according to the terminology of [18]), from which it follows that this queue object can be wait-free implemented from atomic registers and consensus objects [18]. Now, a process that wants to obtain a new name does the following: (1) it deposits its initial name in the queue, (2) then reads the content of the queue, and finally (3) takes as its new name its position in the sequence of initial names read from queue. It is easy to see that if p processes participate, they obtain the new names from 1 to p , which means that consensus objects are powerful enough to obtain the smallest possible new name space.

The aim of the paper is to try circumventing the lower bound $M = 2p - 1$ associated with the adaptive wait-free renaming problem, by enriching the underlying read/write register system with appropriate objects. More precisely, given M with $p \leq M \leq 2p - 1$, which objects (when added to a read/write register system) allow designing an M -renaming wait-free algorithm (without allowing designing an $(M - 1)$ -renaming algorithm). The previous discussion on consensus objects suggests to investigate k -set agreement objects to attain this goal, and to study the tradeoff relating the value of k with the new renaming space. The *k-set agreement* problem is a distributed coordination problem (k defines the coordination degree it provides the processes with) that generalizes the consensus problem: each process proposes a value, and any process that does not crash must decide a value in such a way that at most k distinct values are decided and any decided value is a proposed value. The smaller the coordination degree k , the more coordination imposed on the participating processes: $k = 1$ is the more constrained version of the problem (it is consensus), while $k = n$ means no coordination at all.

From k -set to $(2p - \lceil \frac{p}{k} \rceil)$ -renaming Assuming k -set agreement base objects, and $p \leq n$ participating processes, the paper presents an adaptive wait-free renaming algorithm providing a renaming space whose size is $M = (2p - \lceil \frac{p}{k} \rceil)$. Interestingly, when considering the two extreme cases we have the following: $k = 1$ (consensus) gives $M = p$ (the best that can be attained), while $k = n$ (no additional coordination power) gives $M = 2p - 1$, meeting the lower bound for adaptive renaming in read/write register systems.

The proposed algorithm follows Gafni’s reduction style [13]. It is inspired by the adaptive renaming algorithm proposed by Borowsky and Gafni in [9]. In addition to k -set objects, it also uses simple variants of base objects introduced in [9, 10, 15, 16], namely, *strong k-set agreement* [10], *k-participating set* [9,

15, 16]. These objects can be incrementally built from base k -set objects as indicated in Figure 1 (an arrow means “used by”, the reverse direction means “can be reduced to”).



Figure 1: From k -set to $(2p - \lfloor \frac{p}{k} \rfloor)$ -renaming

The renaming algorithm is surprisingly simple. It is based on a very well-known basic strategy: decompose a problem into independent subproblems, solve each subproblem separately, and finally piece together the subproblem results to produce the final result. More precisely, the algorithm proceeds as follows:

- Using a k -participating set object, the processes are partitioned into independent subsets of size at most k .
- In each partition, the processes compete in order to acquire new names from a small name space. Let h be the number of processes that belong to a given partition. They obtain new names in the space $[1..2h - 1]$.
- Finally, the name spaces of all the partitions are concatenated in order to obtain a single name space $[1..M]$.

The key of the algorithm is the way it uses a k -participating set object to partition the p participating processes in such a way that, when the new names allocated in each partition are pieced together, the new name space is upper bounded by $M = (2p - \lfloor \frac{p}{k} \rfloor)$ ⁽¹⁾. Interestingly, the processes that belong to the same partition can use any wait-free adaptive renaming algorithm to obtain new names within their partition (distinct partitions can even use different algorithms). This noteworthy modularity property adds a generic dimension to the proposed algorithm.

From the oracle Ω^k to k -set objects Unfortunately, k -set agreement objects cannot be wait-free implemented from atomic registers [10, 19, 23]. So, the paper investigates additional equipment the asynchronous read/write register system has to be enriched with in order k -set agreement objects can be implemented. To that aim, the paper investigates a family of leader oracles (denoted here $(\Omega^z)_{1 \leq z \leq n}$), and presents a k -set algorithm based on read/write registers and any oracle of such a class Ω^k .

So, the paper provides reductions showing that adaptive wait-free $(2p - \lfloor \frac{p}{k} \rfloor)$ -renaming can be reduced to the Ω^k leader oracle class. To our knowledge, this is the first time that oracles (failure detectors) are proposed and used to circumvent the $2p - 1$ adaptive renaming space lower bound. Several problems remain open. The most crucial is the statement of the minimal information on process crashes that are necessary and sufficient for bypassing the lower bound $2p - 1$. This seems to be related to the open problem that consists in finding the minimal assumptions on failures that allow solving the k -set agreement problem.

Roadmap The paper is made up of 6 sections. Section 2 presents the asynchronous computation model. Then, Section 3 describes the adaptive renaming algorithm. This algorithm is based on a k -participating set object. Section 4 visits Gafni’s reduction land by showing how the k -participating set object can be built

¹When we were designing that algorithm, we had in mind sequential sorting algorithms such as *quicksort*, *mergesort* and *heapsort*, and were thinking to possible relations linking renaming and sorting.

from a k -set object. Then, Section 5 describes an algorithm that constructs a k -set object in an asynchronous read/write system equipped with a leader oracle of the class Ω^k . Finally, Section 6 provides a few concluding remarks while presenting open problems.

2 Asynchronous system model

Process model The system consists of n processes that we denote p_1, \dots, p_n . The integer i is the index of p_i . Each process p_i has an initial name id_i such that $id_i \in [1..N]$. Moreover, a process does not know the initial names of the other processes; it only knows that no two processes have the same initial name. A process can crash. Given an execution, a process that crashes is said to be *faulty*, otherwise it is *correct* in that execution. Each process progresses at its own speed, which means that the system is asynchronous.

Coordination model The processes cooperate and communicate through two types of reliable objects: atomic multi-reader/single-write registers, and k -set objects.

A k -set object KS provides the processes with a single operation denoted `kset_proposek()`. It is a one-shot object in the sense that each process can invoke `KS.kset_proposek()` at most once. When a process p_i invokes `KS.kset_proposek(v)`, we say that it “proposes v ” to the k -set object KS . If p_i does not crash during that invocation, it obtains a value v' (we then say “ p_i decides v' ”). A k -set object guarantees the following two properties: a decided value is a proposed value, and no more than k distinct values are decided.

Notation Identifiers with upper case letters are used to denote shared registers or shared objects. Lower case letters are used to denote local variables; in that case the process index appears as a subscript. As an example, $level_i[j]$ is a local variable of the process p_i , while $LEVEL[j]$ is an atomic register.

3 An adaptive $(2p - \lceil \frac{p}{k} \rceil)$ -renaming algorithm

This section presents an adaptive wait-free $(2p - \lceil \frac{p}{k} \rceil)$ -renaming algorithm (where p is the number of processes that participate in the algorithm). As announced previously, this algorithm is based on atomic registers and k -set objects.

3.1 Non-triviality

Let us observe that the trivial renaming algorithm where p_i takes i as its new name is not adaptive, as the renaming space would always be $[1..m]$, where m is the greatest index of a participating process (as an example consider the case where only p_1 and p_n are participating in the renaming). To rule out this type of ineffective solution, we consider the following requirement for a renaming algorithm [7]:

- The code executed by process p_i with initial name id is exactly the same as the code executed by process p_j with initial name id .

This constraint imposes a form of anonymity with respect to the process initial names. It also means that there is a strong distinction between the index i associated with p_i and its original name id_i . The initial name id_i can be seen as a particular value defined in p_i 's initial context. Differently, the index i can be seen as a pointer to the atomic registers that can be written only by p_i . This means that the indexes define the underlying “communication infrastructure”.

3.2 k -participating set object

The renaming algorithm is based on a k -participating set object. Such an object generalizes the *participating set* object first defined in [9]. The particular case $k = 2$ when $n = 3$ has been introduced in [15, 16].

Definition A k -participating set object PS is a one-shot object that provides the processes with a single operation denoted `participating_setk()`. A process p_i invokes that operation with its name id_i as a parameter. The invocation $PS.\text{participating_set}_k(id_i)$ returns a set S_i to p_i (if p_i does not crash while executing that operation). The semantics of the object is defined by the following properties [9, 15, 16]:

- Self-membership: $\forall i: id_i \in S_i$.
- Comparability: $\forall i, j: S_i \subseteq S_j \vee S_j \subseteq S_i$.
- Immediacy: $\forall i, j: (id_i \in S_j) \Rightarrow (S_i \subseteq S_j)$.
- Bounded simultaneity: $\forall \ell: 1 \leq \ell \leq n: |\{j : |S_j| = \ell\}| \leq k$.

The set S_i obtained by a process p_i can be seen as the set of processes that, from its point of view, have accessed or are accessing the k -participating set object. A process always sees itself (self-membership). Moreover, such an object guarantees that the S_i sets returned to the process invocations can be totally ordered by inclusion (comparability). Additionally, this total order is not at all arbitrary: it ensures that, if p_j sees p_i (i.e., $id_i \in S_j$) it also sees all the processes seen by p_i (Immediacy). As a consequence if $id_i \in S_j$ and $id_j \in S_i$, we have $S_i = S_j$. Finally, the object guarantees that no more than k processes see the same set of processes (Bounded simultaneity).

As we will see later (Section 3.2), such an object can be constructed from k -set objects. When $k = n$, the bounded simultaneity requirement is always satisfied, and can consequently be omitted (then, the definition boils down to the participating set definition introduced in [9]).

level	stopped processes	S_i sets
10	p_5, p_9	$S_5 = S_9 = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}\}$
9		empty level
8	p_1, p_3, p_{10}	$S_1 = S_3 = S_{10} = \{p_1, p_2, p_3, p_4, p_6, p_7, p_8, p_{10}\}$
7		empty level
6		empty level
5	p_2, p_8	$S_2 = S_8 = \{p_2, p_4, p_6, p_7, p_8\}$
4		empty level
3	p_7	$S_7 = \{p_4, p_6, p_7\}$
2	p_4, p_6	$S_4 = S_6 = \{p_4, p_6\}$
1		empty level

Table 1: An example of k -participating object ($p = 10 \leq n, k = 3$)

Notation and properties Let S_j be the set returned to p_j after it has invoked `participating_setk(idj)`, and $\ell = |S_j|$ (notice that $0 \leq \ell \leq n$). The integer ℓ is called the *level* of p_j , and we say “ p_j is -or stopped- at level ℓ ”. If there is a process p_j such that $|S_j| = \ell$, we say “the level ℓ is not empty”, otherwise we say “the level ℓ is empty”. Let \mathcal{L} be the set of non-empty levels ℓ , $|\mathcal{L}| = m \leq n$. Let us order the m levels of \mathcal{L} according to their values, i.e., $\ell_1 < \ell_2 < \dots < \ell_m$ (this means that the levels in $\{1, \dots, n\} \setminus \{\ell_1, \dots, \ell_m\}$ are empty).

$|S_j| = \ell$ (p_j stopped at level ℓ) means that, from p_j point of view, there are exactly ℓ processes that (if they do not crash) stop at the levels ℓ' such that $1 \leq \ell' \leq \ell$. Moreover, these processes are the processes that define S_j . (It is possible that some of them have crashed before stopping at a level, but this fact cannot be known by p_j .) We have the following properties:

- If p processes invoke `participating_setk(ℓ)`, no process stops at a level higher than p .
- $(|S_i| = |S_j| = \ell) \Rightarrow (S_i = S_j)$ (from the comparability property).
- Let S_i and S_j such that $|S_i| = \ell_x$ and $|S_j| = \ell_y$ with $x < y$.
 - $S_i \subset S_j$ (from $\ell_x < \ell_y$, and the comparability property).
 - $|S_j \setminus S_i| = |S_j| - |S_i| = \ell_y - \ell_x$ (consequence of the set inclusion $S_i \subset S_j$).

A k -participating set object can be seen as “spreading” the $p \leq n$ participating processes on at most p levels ℓ . This spreading is such that (1) there are at most k processes per level, and (2) each process has a consistent view of the spreading (where “consistent” is defined by the self-membership, comparability and immediacy properties). As an example, let us consider Table 1 that depicts the sets S_i returned to $p = 10$ processes participating in a k -participating set object (with $k = 3$), in a failure-free run. As we can see some levels are empty. Two processes, p_2 and p_8 , stopped at level 5; their sets are equal and contain exactly five processes, namely the processes that stopped at a level ≤ 5 .

The following lemma captures an important property provided by a k -participating set object. Let $ST[\ell_x] = \{j \text{ such that } |S_j| = \ell_x\}$ (the processes that have stopped at the level ℓ_x). For consistency purpose, let $\ell_0 = 0$.

Lemma 1 $|ST[\ell_x]| \leq \min(k, \ell_x - \ell_{x-1})$.

Proof $|ST[\ell_x]| \leq k$ follows immediately from the bounded simultaneity property. To show $|ST[\ell_x]| \leq \ell_x - \ell_{x-1}$, let us consider two processes p_j and p_i such that p_j stops at the level ℓ_x while p_i stops at the level ℓ_{x-1} . We have:

1. $|S_j| = \ell_x$ and $|S_i| = \ell_{x-1}$ (definition of “a process stops at a level”).
2. $ST[\ell_x] \subseteq S_j$ (from the self-membership and comparability properties),
3. $ST[\ell_x] \cap S_i = \emptyset$ (from $S_j \neq S_i$ and the immediacy and self-membership properties),
4. $ST[\ell_x] \subseteq S_j \setminus S_i$ (from the items 2 and 3),
5. $|S_j \setminus S_i| = \ell_x - \ell_{x-1}$ (previous discussion),
6. $|ST[\ell_x]| \leq \ell_x - \ell_{x-1}$ (from the items 4 and 5).

□_{Lemma 1}

Considering again Table 1, let us assume that the processes p_1 , p_3 and p_{10} have crashed while they are at level $\ell = 8$, and before determining their sets S_1 , S_3 and S_{10} . The level $\ell = 8$ is now empty (as no process stops at that level), and the levels 10 and 5 are now consecutive non-empty levels. We have then $ST[10] = \{p_5, p_9\}$, $ST[5] = \{p_2, p_8\}$, and $|ST[10]| = 2 \leq \min(k, 10 - 5)$.

3.3 An adaptive renaming protocol

The adaptive renaming algorithm is described in Figure 2. When a process p_i wants to acquire a new name, it invokes $\text{new_name}(id_i)$. It then obtains a new name when it executes line 05. Remind that p denotes the number of processes that participate in the algorithm.

Base objects The algorithm uses a k -participating set object denoted PS , and a size n array of adaptive renaming objects, denoted $RN[1..n]$.

Each base renaming object $RN[y]$ can be accessed by at most k processes. It provides them with an operation denoted $\text{rename}()$. When accessed by $h \leq k$ processes, it allows them to acquire new names within the renaming space $[1..2h - 1]$. Interestingly, such adaptive wait-free renaming objects can be built from atomic registers, e.g., [2, 4, 9] (for completeness, one of them is described in appendix A). As noticed in the introduction, this feature provides the proposed algorithm with a modularity dimension as $RN[y]$ and $RN[y']$ can be implemented differently.

The algorithm: principles and description The algorithm is based on the following (well-known) principle.

- Part 1. Divide for conquer.

A process p_i first invokes $PS.\text{participating_set}_k(id_i)$ to obtain a set S_i satisfying the self-membership, comparability, immediacy and bounded simultaneity properties (line 01). It follows from these properties that (1) at most k processes obtain the same set S (and consequently belong to the same partition), and (2) there are at most p distinct partitions.

An easy and unambiguous way to identify the partition p_i belongs to is to consider the level at which p_i stopped in the k -participating set object, namely, the level $\ell = |S_i|$. The $h \leq k$ processes in the partition $\ell = |S_i|$ compete then among themselves to acquire a new name. This is done by p_i invoking the appropriate renaming object, i.e., $RN[|S_i|].\text{rename}(id_i)$ (line 03). As indicated before, these processes obtain new names in renaming space $[1..2h - 1]$.

<p>operation $\text{new_name}(id_i)$:</p> <p>(01) $S_i \leftarrow PS.\text{participating_set}_k(id_i)$;</p> <p>(02) $base_i \leftarrow (2 \times S_i - \lceil \frac{ S_i }{k} \rceil)$;</p> <p>(03) $offset_i \leftarrow RN[S_i].\text{rename}(id_i)$;</p> <p>(04) $myname_i \leftarrow base_i - offset_i + 1$;</p> <p>(05) $\text{return}(myname_i)$</p>
--

Figure 2: Generic adaptive renaming algorithm (code for p_i)

- Part 2. Piece together the results of the subproblems.

The final name assignment is done according to very classical ($base, offset$) rule. A base is attributed to each partition as follows. The partition $\ell = |S_i|$ is attributed the base $2 \times |S_i| - \lceil \frac{|S_i|}{k} \rceil$ (line 02). Let us notice that no two partitions are attributed the same base. Then, a process p_i in partition ℓ considers the new name obtained from $RN[\ell]$ as an offset (notice that an offset is never equal to 0). It determines its final new name from the base and offset values it has been provided with, considering the name space starting from the base and going down (line 04).

3.4 Proof of the algorithm

Lemma 2 *The algorithm described in Figure 2 ensures that no two processes obtain the same new name.*

Proof Let p_i be a process such that $|S_i| = \ell_x$. That process is one of the $|ST[\ell_x]|$ processes that stop at the level ℓ_x and consequently use the underlying renaming object $RN[\ell_x]$. Due to the property of that renaming object, p_i computes a value $offset_i$ such that $1 \leq offset_i \leq 2 \times |ST[\ell_x]| - 1$. Moreover, as $|ST[\ell_x]| \leq \min(k, \ell_x - \ell_{x-1})$ (Lemma 1), the previous relation becomes $1 \leq offset_i \leq 2 \times \min(k, \ell_x - \ell_{x-1})$.

On another side, the renaming space attributed to the processes p_i of $ST[\ell_x]$ starts at the base $2\ell_x - \lceil \frac{\ell_x}{k} \rceil$ (included) and goes down until $2\ell_{x-1} - \lceil \frac{\ell_{x-1}}{k} \rceil$ (excluded). Hence the size of this renaming space is

$$2(\ell_x - \ell_{x-1}) - \left(\lceil \frac{\ell_x}{k} \rceil - \lceil \frac{\ell_{x-1}}{k} \rceil \right).$$

It follows from these observations that a sufficient condition for preventing conflict in name assignment is to have

$$2 \times \min(k, \ell_x - \ell_{x-1}) - 1 \leq 2(\ell_x - \ell_{x-1}) - \left(\lceil \frac{\ell_x}{k} \rceil - \lceil \frac{\ell_{x-1}}{k} \rceil \right).$$

We prove that the algorithm satisfies the previous relation by considering two cases according to the minimum between k and $\ell_x - \ell_{x-1}$. Let

$$\ell_x = q_x k + r_x \text{ with } 0 \leq r_x < k \quad (\text{i.e., } \lceil \frac{r_x}{k} \rceil \in \{0, 1\}), \quad \text{and}$$

$$\ell_{x-1} = q_{x-1} k + r_{x-1} \text{ with } 0 \leq r_{x-1} < k \quad (\text{i.e., } \lceil \frac{r_{x-1}}{k} \rceil \in \{0, 1\}), \quad \text{from which we have}$$

$$\ell_x - \ell_{x-1} = (q_x - q_{x-1}) k + (r_x - r_{x-1}).$$

- Case $\ell_x - \ell_{x-1} \leq k$.

In that case, the relation to prove simplifies and becomes $\lceil \frac{\ell_x}{k} \rceil - \lceil \frac{\ell_{x-1}}{k} \rceil \leq 1$, i.e., $(q_x + \lceil \frac{r_x}{k} \rceil) - (q_{x-1} + \lceil \frac{r_{x-1}}{k} \rceil) \leq 1$, that can be rewritten as $(q_x - q_{x-1}) + (\lceil \frac{r_x}{k} \rceil - \lceil \frac{r_{x-1}}{k} \rceil) \leq 1$.

Moreover, from $\ell_x - \ell_{x-1} = (q_x - q_{x-1}) k + (r_x - r_{x-1})$ and $\ell_x - \ell_{x-1} \leq k$, we have $(q_x - q_{x-1}) k + (r_x - r_{x-1}) \leq k$ from which we can extract two subcases:

- Case $q_x - q_{x-1} = 1$ and $r_x = r_{x-1}$.

In that case, it trivially follows from the previous formulas that $(q_x - q_{x-1}) + (\lceil \frac{r_x}{k} \rceil - \lceil \frac{r_{x-1}}{k} \rceil) \leq 1$, which proves the lemma for that case.

- Case $q_x = q_{x-1}$ and $0 \leq r_x - r_{x-1} \leq k$.

In that case we have to prove $\lceil \frac{r_x}{k} \rceil - \lceil \frac{r_{x-1}}{k} \rceil \leq 1$. As $\lceil \frac{r_x}{k} \rceil, \lceil \frac{r_{x-1}}{k} \rceil \in \{0, 1\}$, we have $\lceil \frac{r_x}{k} \rceil - \lceil \frac{r_{x-1}}{k} \rceil \leq 1$, which proves the lemma for that case.

- Case $k < \ell_x - \ell_{x-1}$.

After simple algebraic manipulations, the formula to prove becomes:

$$(2k - 1)(q_x - q_{x-1} - 1) + 2(r_x - r_{x-1}) - \left(\lceil \frac{r_x}{k} \rceil - \lceil \frac{r_{x-1}}{k} \rceil \right) \geq 0.$$

Moreover, we have now $\ell_x - \ell_{x-1} = (q_x - q_{x-1}) k + (r_x - r_{x-1}) > k$, from which, as $0 \leq r_x, r_{x-1} < k$, we can conclude $q_x - q_{x-1} \geq 1$. We consider two cases.

- $q_x - q_{x-1} = 1$.

The formula to prove becomes $2(r_x - r_{x-1}) \geq \lceil \frac{r_x}{k} \rceil - \lceil \frac{r_{x-1}}{k} \rceil$.

From $\ell_x - \ell_{x-1} > k$ we have:

$$* \ r_x > r_{x-1}, \text{ from which (as } r_x \text{ and } r_{x-1} \text{ are integers) we conclude } 2(r_x - r_{x-1}) \geq 2.$$

* $1 \geq \lceil \frac{r_x}{k} \rceil \geq \lceil \frac{r_{x-1}}{k} \rceil \geq 0$, from which we conclude $\lceil \frac{r_x}{k} \rceil - \lceil \frac{r_{x-1}}{k} \rceil \leq 1$.

By combining the previous relations we obtain $2 \geq 1$ which proves the lemma for that case.

– $q_x - q_{x-1} > 1$. Let $q_x - q_{x-1} = 1 + \alpha$ (where α is an integer ≥ 1).

The formula to prove becomes

$$(2k - 1)\alpha + 2(r_x - r_{x-1}) - (\lceil \frac{r_x}{k} \rceil - \lceil \frac{r_{x-1}}{k} \rceil) \geq 0.$$

As $0 \leq r_x, r_{x-1} < k$, the smallest value of $r_x - r_{x-1}$ is $-(k - 1)$. Similarly, the greatest value of $\lceil \frac{r_x}{k} \rceil - \lceil \frac{r_{x-1}}{k} \rceil$ is 1.

It follows that, the smallest value of the left side of the formula is $(2k - 1)\alpha - 2(k - 1) - 1 = 2k\alpha - (2k + \alpha) + 1 = (2k - 1)(\alpha - 1)$. As $k \geq 1$ and $\alpha \geq 1$, it follows that the left side is never negative, which proves the lemma for that case.

□*Lemma 2*

Theorem 1 *The algorithm described in Figure 2 is a wait-free adaptive $(2p - \lceil \frac{p}{k} \rceil)$ -renaming algorithm (where $p \leq n$ is the number of processes that participate in the algorithm).*

Proof The fact that the algorithm is wait-free is an immediate consequence of the fact that base k -set participating set object and the base renaming objects are wait-free. The fact that no two processes obtain the same new name is established in Lemma 2.

If p processes participate in the algorithm, the highest level at which a process stops is p (this follows from the properties of the k -set participating set object). Consequently, the largest base that is used (line O2) is $2p - \lceil \frac{p}{k} \rceil$, which establishes the upper bound on the renaming space. □*Theorem 1*

4 Visiting Gafni’s land: From k -set to k -participating set

This section presents a wait-free transformation from a k -set agreement object to a k -participating set object. It can be seen as a guided visit to Gafni’s reduction land [9, 10, 15, 16].

Let us remind that a k -set object provides each process with an operation $\text{kset_propose}_k()$ that allows it to propose and decide a value in such a way that at most k different values are decided and any decided value is a value that has been proposed by a process. The construction proceeds in two steps: first from k -set agreement to strong k -set agreement, and then from strong k -set agreement to k -participating set.

4.1 From set agreement to strong set agreement

Let us observe that, given a k -set object, it is possible that no process decides the value it has proposed. This feature is the “added value” provided by a *strong k -set agreement* object: it is a k -set object such that at least one process decides the value it has proposed [10]. The corresponding operation is denoted $\text{strong_kset_propose}_k()$.

In addition to a k -set object KS , the processes cooperate by accessing an array $DEC[1..n]$ of one-writer/multi-reader atomic registers. That array is initialized to $[\perp, \dots, \perp]$. $DEC[i]$ can be written only by p_i . The array is provided with a $\text{snapshot}()$ operation. Such an operation returns a value of the whole array as if that value has been obtained by atomically reading the whole array [1]. Let us remind that such an operation can be wait-free implemented on top of atomic read/write base registers (the best snapshot

```

operation strong_kset_proposek(idi) :
(01)  DEC[i] ← KS.kset_proposek(idi);
(02)  deci[1..n] ← snapshot(DEC[1..n]);
(03)  if (∃h : deci[h] = idi) then decisioni ← idi else decisioni ← deci[i] end_if;
(04)  return(decisioni)

```

Figure 3: Strong k -set agreement algorithm (code for p_i)

algorithm known so far costs $O(n \log n)$ atomic register accesses [6]). This means that the base write operations (on each array entry) and the snapshot operations are linearizable [18].

The construction (introduced in [10]) is described in Figure 3. A process p_i first proposes its original name to the underlying k -set object KS , and writes the value it obtains (an original name) into $DEC[i]$ (line 01). Then, p_i atomically reads the whole array (line 02). Finally, if it observes that some process has decided its original name id_i , p_i also decides id_i , otherwise p_i decides the original name it has been provided with by the k -set object (lines 03-04).

Theorem 2 [10] *The algorithm described in Figure 3 wait-free implements a strong k -set agreement object.*

Proof Let us first observe that it trivially follows from the algorithm text that no process returns a name that has not been decided by the k -set object KS . So, only names of participating processes are decided. It follows that the values decided from the strong k -set object SKS satisfy the k -set agreement properties.

If a process p_i , whose original name is one of the names decided by the k -set object, crashes before returning at line 04, it is always possible to consider that p_i would have returned its name at line 04 and crashed just after, which proves the theorem. So, let us consider that none of the processes, whose original name has been decided by the k -set object KS , crashes. If one of these processes p_i is such that the predicate $(\exists h : dec_i[h] = id_i)$ is true when p_i evaluates it, the theorem follows.

So, let us suppose that no process p_i (whose original name is decided by the k -set object) crashes or finds the predicate $(\exists h : dec_i[h] = id_i)$ satisfied when it evaluates it. There is consequently a cycle $j_1, j_2, \dots, j_x, j_1$ on a subset of these processes defined as follows: $id_{j_2} = DEC[j_1]$, $id_{j_3} = DEC[j_2]$, \dots , $id_{j_1} = DEC[j_x]$. Among the processes of this cycle, let us consider the process p_{j_α} that is the last to update its entry $DEC[j_\alpha]$, thereby creating the cycle. (Let us observe that, as the write and snapshot operations that access the array DEC are linearizable, such a “last” process p_{j_α} does exist.) But then, when p_{j_α} executes line 03, the predicate $(\exists h : dec_{j_\alpha}[h] = id_{j_\alpha})$ is necessarily true (as p_{j_α} completes the cycle and -due to the snapshot operation- sees that cycle). It follows that p_{j_α} decides its own original name at line 03-04, which proves the theorem. $\square_{\text{Theorem 2}}$

4.2 From strong set agreement to k -participating set

The specification of a k -participating set object has been defined in Section 3.2. The present section shows how such an object PS can be wait-free implemented from an array of strong k -set agreement objects; this array is denoted $SKS[1..n]$. This construction generalizes the construction proposed in [15, 16] that considers $n = 3$ and $k = 2$. In addition to the array $SKS[1..N]$ of strong k -set agreement objects, the construction uses an array of one-writer/multi-reader atomic registers denoted $LEVEL[1..n]$. As before only p_i can write $LEVEL[i]$. The array is initialized to $[n + 1, \dots, n + 1]$.

The algorithm is based on what we call *Borowski-Gafni’s ladder*, a wait-free object strong k -set agreement introduced in [9]. It combines such a ladder object with a k -set agreement object in order to guarantee that no more than k processes, that do not crash, stop at the same step of the ladder.

Borowsky-Gafni's ladder Let us consider the array $LEVEL[1..n]$ as a ladder. Initially, a process is at the top of the ladder, namely, at level $n + 1$. Then it descends the ladder, one step after the other, according to predefined rules until it stops at some level (or crashes). While descending the ladder, a process p_i registers its current position in the ladder in the atomic register $LEVEL[i]$.

After it has stepped down from one ladder level to the next one, a process p_i computes a local view (denoted $view_i$) of the progress of the other processes in their descent of the ladder. That view contains the processes p_j seen by p_i at the same or a lower ladder level (i.e., such that $level_i[j] \leq LEVEL[i]$). Then, if the current level ℓ of p_i is such that p_i sees at least ℓ processes in its view (i.e., processes that are at its level or a lower level) it stops at the level ℓ of the ladder. This behavior is described by the following algorithm [9]:

```

repeat  $LEVEL[i] \leftarrow LEVEL[i] - 1$ ;
    for  $j \in \{1, \dots, n\}$  do  $level_i[j] \leftarrow LEVEL[j]$  end_do;
     $view_i \leftarrow \{j : level_i[j] \leq LEVEL[i]\}$ ;
until  $(|view_i| \geq LEVEL[i])$  end_repeat;
let  $S_i = view_i$ ; return  $(S_i)$ .

```

This very elegant algorithm satisfies the following properties [9]. The sets S_i of the processes that terminate the algorithm, satisfy the self-membership, comparability and immediacy properties of the k -participating set object. Moreover, if $|S_i| = \ell$, then p_i stopped at the level ℓ , and there are ℓ processes whose current level is $\leq \ell$.

From a ladder to a k -participating set object The construction, described in Figure 4, is nearly the same as the construction given in [15, 16]. It uses the previous ladder algorithm as a skeleton to implement a k -participating set object. When it invokes $participating_set_k()$, a process p_i provides its original name as input parameter. This name will be used by the underlying strong k -participating set object. The array $INIT_NAME[1..n]$ is initialized to $[\perp, \dots, \perp]$. $INIT_NAME[i]$ can be written only by p_i .

<pre> operation $participating_set_k(id_i)$ (01) $INIT_NAME[i] \leftarrow id_i$; (02) repeat $LEVEL[i] \leftarrow LEVEL[i] - 1$; (03) for $j \in \{1, \dots, n\}$ do $level_i[j] \leftarrow LEVEL[j]$ end_do; (04) $view_i \leftarrow \{j : level_i[j] \leq LEVEL[i]\}$; (05) if $(LEVEL[i] > k) \wedge (view_i = LEVEL[i])$ (06) then let $\ell = LEVEL[i]$; (07) $ans_i \leftarrow SKS[\ell].strong_kset_propose_k(id_i)$; (08) $ok_i \leftarrow (ans_i = id_i)$ (09) else $ok_i \leftarrow true$ (10) end_if (11) until $(view_i \geq LEVEL[i]) \wedge ok_i$ end_repeat; (12) let $S_i = \{id \mid \exists j \in view_i \text{ such that } INIT_NAME[j] = id\}$; (13) return (S_i) </pre>

Figure 4: k -participating set algorithm (code for p_i)

If, in the original Borowski-Gafni's ladder, a process p_i stops at a ladder level $\ell \leq k$, it can also stop at the same level in the k -set participating object. This follows from the fact that, as $|view_i| = \ell \leq k$ when p_i stops descending, we know from the ladder properties that at most $\ell \leq k$ processes are at the level ℓ (or at a lower level). So, when $LEVEL[i] \leq k$ (line 05), p_i sets ok_i to $true$ (line 05). It consequently exits the repeat loop (line 11) and we can affirm that no more than k processes do the same, thereby satisfying the bounded simultaneity property.

So, the main issue of the algorithm is to satisfy the bounded simultaneity property when the level ℓ at

which p_i should stop in the original Borowski-Gafni's ladder is higher than k . In that case, p_i uses the underlying strong k -set agreement object $SKS[\ell]$ to know if it can stop at that level (lines 07-08). This k -participating set object ensures that at least one (and at most k) among the participating processes that should stop at level ℓ in the original Borowski-Gafni's ladder, do actually stop. If a process p_i is not allowed to stop (we have then $ok_i = false$ at line 08), it is required to descend to the next step of the ladder (lines 11 and 01). When a process stops at a level ℓ , there are exactly ℓ processes at the levels $\ell' \leq \ell$. This property is maintained when a process steps down from ℓ to $\ell - 1$ (this follows from the fact that when a process is required to step down from $\ell > k$ to $\ell - 1$ because $\ell > k$, at least one process remains at the level ℓ due to the k -set agreement object $SKS[\ell]$).

5 From Ω^k to k -set

This section shows that a k -set object can be built in a single-writer/multi-reader atomic register system, equipped with an oracle (failure detector) of the class Ω^k .

5.1 The oracle class Ω^k

A family of oracle classes $(\Omega^z)_{1 \leq z \leq n}$ has been introduced in [22]. That definition implicitly assumes that all the processes are participating. We extend here this definition by making explicit the notion of participating processes. More precisely, an oracle of the class Ω^z provides the processes with an operation denoted $leader()$ that satisfies the following properties:

- Output size: each time it is invoked, $leader()$ provides the invoking process with a set of at most z *participating* process identities (e.g., $\{id_{x_1}, \dots, id_{x_z}\}$).
- Eventual multiple leadership: There is a time after which all the $leader()$ invocations return forever the same set. Moreover, this set includes at least one *correct participating* process (if any).

It is important to notice that each instance of Ω^k is defined with respect to the *context* where it is used. This context is defined by the participating processes. This means that if Ω^k is used to construct a given object, say a k -set object KS , the participating processes for that failure detector instance are the processes that invoke $KS.kset_propose_k()$. Let us remark that, during an arbitrary long period, the participating processes that invoke $leader()$ can see different sets of leaders, and no process knows when this “anarchy” period is over. It is also possible that some of the processes that are eventually elected as permanent leaders, are faulty processes.

When all the processes are assumed to participate, Ω^1 is nothing else than the leader failure detector denoted Ω introduced in [12], where it is shown that it is the weakest failure detector for solving the consensus problem in asynchronous systems².

5.2 From Ω^k to k -set agreement

In addition to an oracle of the class Ω^k , the proposed k -set agreement algorithm is based on a variant, denoted KA , of a round-based object introduced in [17] to capture the safety properties of Paxos-like consensus algorithms [14, 20]. The leader oracle is used to ensure the liveness of the algorithm. KA is used to abstract away the safety properties of the k -set problem, namely, at most k values are decided, and the decided values are have been proposed.

²So, the lower bound proved in [12], on the power of failure detectors, assumes that all the correct processes are participating.

The KA object This object provides the processes with an operation denoted $\text{alpha_propose}_k(v_i)$. That operation has two input parameters: the value v_i proposed by the invoking process p_i (here its name id_i), and a round number (that allows identifying the invocations). The KA object assumes that no two processes use the same round numbers, and successive invocations by the same process use increasing round numbers. Given a KA object, the invocations $\text{alpha_propose}_k()$ satisfy the following properties:

- **Validity:** the value returned by any invocation $\text{alpha_propose}_k()$ is a proposed value or \perp .
- **Agreement:** At most k different non- \perp values can be returned by the whole set of $\text{alpha_propose}_k()$ invocations.
- **Convergence:** If there is a time after which the operation $\text{alpha_propose}_k()$ is invoked infinitely often, and these invocations are issued by an (unknown but fixed) set of at most k processes, then there is a time after which none of these invocations returns \perp .

The k -set algorithm The algorithm constructing a k -set object KS (accessed by at most n processes³), is described in Figure 5. As in previous algorithms, it uses an array $DEC[1..n]$ of one-writer/multi-reader atomic registers. Only p_i can write $DEC[i]$. The array is initialized to $[\perp, \dots, \perp]$. The algorithm is very simple. If a value has already been decided ($\exists j : DEC[j] \neq \perp$), p_i decides it. Otherwise, p_i looks if it is a leader. If it is not, it loops. If it is a leader ($id_i \in \text{leader}()$), p_i invokes $\text{alpha_propose}_k(r_i, v_i)$ and writes in $DEC[i]$ the value it obtains (it follows from the specification of KA that that value it writes is \perp or a proposed value).

```

operation kset_proposek(vi):
(01)  ri ← (i - n);
(02)  while (∃j : DEC[j] = ⊥) do
(03)    if (idi ∈ leader()) then ri ← ri + n; DEC[i] ← KA.alpha_proposek(ri, vi) end_if
(04)  end_while;
(05)  let decidedi = any DEC[j] ≠ ⊥;
(06)  return(decidedi)

```

Figure 5: An Ω^k -based k -set algorithm (code for p_i)

It is easy to see that no two processes use the same round numbers, and each process uses increasing round numbers. It follows directly from the agreement property of the KA object, that at any time, the array $DEC[1..n]$ contains at most k values different from \perp . Moreover, due the validity property of KA , these values have been proposed.

It is easy to see that, as soon as a process has written a non- \perp value in $DEC[1..n]$, any $\text{kset_propose}(v_i)$ invocation issued by a correct process terminates. So, in order to show that the algorithm is wait-free, we have to show that at least one process writes a non- \perp value in $DEC[1..n]$. Let us assume that no process deposits a value in this array. Due to the eventual multiple leadership property of Ω^k , there is a time τ after which the same set of $k' \leq k$ participating processes are elected as permanent leaders, and this set includes at least one correct process. It follows from the algorithm that, after τ , at most k processes invoke $KA.\text{alpha_propose}_k()$, and one of them is correct. It follows from the convergence property of the KA object, that there is a time $\tau' \geq \tau$ after which no invocation returns \perp . Moreover, as at least one correct process belongs to the set of elected processes, that process eventually obtains a non- \perp value from an invocation, and consequently deposits that non- \perp value in $DEC[1..n]$. The algorithm is consequently wait-free.

³Let us remind that the construction of each $SKS[\ell]$ object used in Figure 4 is based on an underlying k -set object KS object.

5.3 Implementing KA

An algorithm constructing a KA object is described in Figure 6. It uses an array of single-writer/multi-reader atomic registers $REG[1..n]$. As previously, $REG[i]$ can be written only by p_i . A register $REG[i]$ is made up of three fields $REG[i].lre$, $REG[i].lrww$ and $REG[i].val$ whose meaning is the following ($REG[i]$ is initialized to $\langle 0, 0, \perp \rangle$):

- $REG[i].lre$ stores the number of the last round entered by p_i . It can be seen as the logical date of the last invocation issued by p_i .
- $REG[i].lrww$ and $REG[i].val$ constitute a pair of related values: $REG[i].lrww$ stores the number of the last round with a write of a value in the field $REG[i].val$. So, $REG[i].lrww$ is the logical date of the last write in $REG[i].val$.

(To simplify the writing of the algorithm, we consider that each field of a register can be written separately. This poses no problem as each register is single writer. A writer can consequently keep a copy of the last value it has written in each register field and rewrite it when that value is not modified.)

```

operation alpha_proposek( $r, v$ ):
(01)   $REG[i].lre \leftarrow r$ ;
(02)  for  $j \in \{1, \dots, n\}$  do  $reg_i[j] \leftarrow REG[j]$  end_do;
(03)  let  $value_i$  be  $reg_i[j].val$  where  $j$  is such that  $\forall x: reg_i[j].lrww \geq reg_i[x].lrww$ ;
(04)  if ( $value_i = \perp$ ) then  $value_i \leftarrow v$  end_if;
(05)   $REG[i].(lrww, v) \leftarrow (r, value_i)$ ;
(06)  for  $j \in \{1, \dots, n\}$  do  $reg_i[j] \leftarrow REG[j]$  end_do;
(07)  if ( $|\{j | reg_i[j].lre \geq r\}| > k$ ) then  $\text{return}(\perp)$ 
(08)  else  $\text{return}(value_i)$  end_if

```

Figure 6: A KA object algorithm (code for p_i)

The principle that underlies the algorithm is very simple: it consists in using a logical time frame (represented here by the round numbers) to timestamp the invocations, and answering \perp when the timestamp of the corresponding invocation does not lie within the k highest dates (registered in $REG[1..n].lre$). To that end, the algorithm proceeds as follows:

- Step 1 (lines 01-02): Access the shared registers.
 - When a process p_i invokes $\text{alpha_propose}_k(r, v)$, it first informs the other processes that the KA object has attained (at least) the date r (line 01). Then p_i reads all the registers in any order (line 02) to know the last values (if any) written by the other processes.
- Step 2 (lines 03-05): Determination and writing of a value.
 - Then, p_i determines a value. In order not to violate the agreement property, it selects the last value (“last” according to the round numbers/logical dates) that has been deposited in a register $REG[j]$. If there is no such value it considers its own value v . After this determination, p_i writes in $REG[i]$ the value it has determined, together with its round number (line 05).
- Step 3 (lines 06-08): Commit or abort.
 - p_i reads again the shared registers to know the progress of the other processes (measured by their round numbers), line 07. If it discovers it is “late”, p_i aborts returning \perp . (Let us observe that this preserves the agreement property.) “To be late” means that the current date r of p_i does not lie within the window defined by the k highest dates (round numbers) currently entered by the processes (these round numbers/dates are registered in the field lre of each entry of the array $REG[1..n]$).

- Otherwise, p_i is not late. It then returns (“commits”) $value_i$ (line 08). Let us observe that, as the notion of “being late” is defined with respect to a window of k dates (round numbers), it is possible that up to k processes are not late and return concurrently up to k distinct non- \perp values.

It directly follows from the code that the algorithm is wait-free. Moreover, in order to expedite the $\alpha_propose_k()$ operation, it is possible to insert the statement

if ($|\{j | reg_i[j].lre \geq r\}| > k$) **then** return(\perp) **end_if**

between the line 02 and the line 03. This allows the invoking process to return \perp when, just after entering the $\alpha_propose_k()$ operation, it discovers it is late.

5.4 Proof of the KA object

Theorem 3 *The algorithm described in Figure 6 wait-free implements a KA object.*

Proof The wait-free property follows directly from the code of the algorithm.

Validity Let us observe that if a value v is written in $REG[i].val$, that value has been previously passed as a parameter in an $\alpha_propose_k()$ invocation. The validity property follows from this observation and the fact that only \perp or a value written in a register $REG[i]$ can be returned from an $\alpha_propose_k()$ invocation.

Convergence Let τ be a time after which there is a set of $k' \leq k$ processes such that each of them invokes $\alpha_propose_k()$ infinitely often. This means that, from τ , the values of $n - k'$ registers $REG[x]$ are no longer modified. Consequently, as the k' processes p_j repeatedly invoke $\alpha_propose_k()$, there is a time $\tau' \geq \tau$ after which each $REG[j].lre$ becomes greater than any $REG[x].lre$ that is no longer modified. There is consequently a time $\tau'' \geq \tau'$ after which the k' processes are such that their registers $REG[j].lre$ contain forever the k greatest timestamp values. It follows from the test done at line 07 that, after τ'' , no $\alpha_propose_k()$ invocation by one of these k' processes can be aborted. Consequently, each of them returns a non- \perp value at line 08.

Agreement If all invocations returns \perp , the agreement property is trivially satisfied. So, let us consider an execution in which at least one $\alpha_propose_k()$ invocation returns a non- \perp value. To prove the agreement property we show that:

- Before the first non- \perp value is returned by an invocation, there is a time at which the algorithm has determined a set V of at least one and at most k non- \perp values⁴.
- Any value $v \neq \perp$ returned by an invocation is a value of V .

To simplify the reasoning, and without loss of generality, we assume that a process that repeatedly invokes $\alpha_propose_k()$, stops invoking that operation as soon as it returns a non- \perp value at line 08.

1. Invariants. $\forall j \in \{1, \dots, n\}$:

- $REG[j].lre$ is increasing (assumption on the successive round numbers used by p_j).
- $REG[j].lrww \leq REG[j].lre$ (because p_j executes line 05 after line 01).

⁴According to the terminology introduced in [11], the set V defines the values that are *locked*. This means that from now on the set of non- \perp values that can be returned is fixed forever: no value outside V can ever be returned.

8. Given an execution, the previous item has extracted a non-empty set V of at most k non- \perp values that can be returned. We now show that (1) from τ , only values of V can be written in $REG[1..n]$ with a timestamp field ($lrww$) greater than r , and (2) a non- \perp value returned by an invocation is necessarily a value of V .

(a) The τ -time invocation issued by a process $p_j \in \bar{L}$ has a round number $REG[j].lre$ that is smaller than $REG[j_1].lre = r$ (this follows from the definition of \bar{L}). As by definition, r is the smallest round number during which a process finds true the predicate of line 07, it follows that any process in \bar{L} needs to issue an invocation with a round number greater than r to have a chance to return a non- \perp value.

(b) Let \mathcal{I}' be the set of all the invocations that have a round number greater than r . They are issued by the processes of \bar{L} or the processes of L whose τ -time invocation has returned \perp at line 07. Let us observe that any invocation of \mathcal{I}' starts after τ .

Let I' be the first invocation of \mathcal{I}' that executes 05. I' (issued by some process p_j) selects (at line 03) a value $value_j$ from a register $REG[y]$ such that $REG[y].lrww \geq REG[j_1].lrww = r$. As up to now, only processes of L have written values in $REG[1..n]$ with a write timestamp ($lrww$) $\geq r$, it follows that I' selects a value from V ⁵. Consequently, this invocation does not add a new value to V .

Let I'' be the invocation of \mathcal{I}' that is the second to execute line 05. The same reasoning (including now I') applies. Etc. It follows from this induction that a value written at line 05 by an invocation of \mathcal{I}' is a value of V , which proves that only values of V can be written in the array $REG[1..n]$ with a write timestamp greater than r .

(c) Finally, an invocation that returns a value at line 08, returns the value it has written at line 05. Due to the definition of r , its round number r' is $\geq r$. It follows that the non- \perp value that is returned is a value of V .

□*Theorem 3*

6 Concluding remarks

What was the paper on This paper has presented a wait-free adaptive renaming algorithm whose renaming space is $M = (2p - \lceil \frac{p}{k} \rceil)$, where p is the number of participating processes. This algorithm relies on an underlying k -set agreement object. It has also been shown how such an object can be built from atomic read/write registers and a leader oracle of the class Ω^k . The construction is based on the reduction style advocated by Gafni [13]. It uses several intermediate objects introduced in [9, 10, 15, 16].

To our knowledge, the proposed construction is the first that uses the (possibly unreliable) information on failures provided by an oracle (failure detector) to circumvent the $2p - 1$ lower bound on the adaptive renaming space. In that sense the paper establishes a connection between Gafni's reductions and failure detectors.

Open problems If $k > t$, there are trivial algorithms for implementing a k -set object in an asynchronous read/write register systems. So, let us assume $k \leq t$. Instead of looking for a wait-free renaming algorithm,

⁵It is possible that, when I' reads the array $REG[1..n]$ at line 02, not all the values of V have yet been written in that array. The important points are here that (1) at least one value of V has already been written in the array (namely, $REG[j_1].val$ with the timestamp $REG[j_1].lrww = r$), and (2) any register $REG[x]$ that currently contains a value not in V , is such that $REG[x].lrww < r$.

we could be interested in a t -resilient adaptive algorithm, i.e., a renaming algorithm that works when the number of crashes does not bypass the model parameter t (the wait-free case being the extreme case $t = n - 1$).

We spent time looking for such an algorithm (without success until now). We nevertheless think that it should be possible to design a t -resilient adaptive M -renaming algorithm from k -set objects, where

$$M = n + (t + 1) - \lceil \frac{t + 1}{k} \rceil.$$

Let us notice that this formula involves the total number of processes n , the resilience bound t , and the parameter k that measures the additional power in presence of crashes -power provided by Ω^k -). When $k > t$ (i.e., when there is no additional power), we obtain $M = n + t$ (that is the lower bound in asynchronous read/write systems).

Another interesting question concerns the implementation of the $\text{alpha_propose}_k()$ operation from a Borowsky-Gafni's ladder-like object. Is it possible? If the answer is "yes", it would shed a new light on the way the safety properties of à la Paxos shared memory consensus algorithms could be implemented.

References

- [1] Afek Y., Attiya H., Dolev D., Gafni E., Merritt M. and Shavit N., Atomic Snapshots of Shared Memory. *Journal of the ACM*, 40(4):873-890, 1993.
- [2] Afek Y. and Merritt M., Fast, Wait-Free $(2k - 1)$ -Renaming. *Proc. 18th ACM Symposium on Principles of Distributed Computing (PODC'99)*, ACM Press, pp. 105-112, Atlanta (GA), 1999.
- [3] Attiya H., Bar-Noy A., Dolev D., Peleg D. and Reischuk R., Renaming in an Asynchronous Environment. *Journal of the ACM*, 37(3):524-548, 1990.
- [4] Attiya H. and Fourn A., Polynomial and Adaptive Long-lived $(2k - 1)$ -Renaming. *Proc. Symposium on Distributed Computing (DISC'00)*, Springer-Verlag LNCS #1914, pp. 149-163, Toledo (Spain), 2000.
- [5] Attiya H. and Fourn A., Adaptive and Efficient Algorithms for Lattice Agreement and Renaming. *SIAM Journal of Computing*, 31(2):642-664, 2001.
- [6] Attiya H. and Rachman O., Atomic Snapshots in $O(n \log n)$ Operations. *SIAM Journal of Computing*, 27(2):319-340, 1998.
- [7] Attiya H. and Welch J., *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, (2d Edition), Wiley-Interscience, 414 pages, 2004.
- [8] Bernstein P.A. and Goodman N., Concurrency Control in Distributed Data Base Systems. *ACM Computing Survey*, 13(2):185-221, 1981.
- [9] Borowsky E. and Gafni E., Immediate Atomic Snapshots and Fast Renaming. *Proc. 12th ACM Symposium on Principles of Distributed Computing (PODC'93)*, ACM Press, pp. 41-51, Ithaca (NY), 1993.
- [10] Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t -Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on Theory of Computation (STOC'93)*, San Diego (CA), pp. 91-100, 1993.
- [11] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, 1996.
- [12] Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685-722, 1996.

- [13] Gafni E., Read/Write Reductions. *DISC/GODEL presentation given as introduction to the 18th Int'l Symposium on Distributed Computing (DISC'04)*, 2004. <http://www.cs.ucla.edu/~eli/eli/godel.ppt>.
- [14] Gafni E. and Lamport L., Disk Paxos. *Distributed Computing*, 16(1):1-20, 2003.
- [15] Gafni E. and Rajsbaum S., Musical Benches. *Proc. 19th Int'l Symposium on Distributed Computing (DISC'05)*, Springer Verlag LNCS #3724, pp. 63–77, 2005.
- [16] Gafni E., Rajsbaum R., Raynal M. and Travers C., The Committee Decision Problem. *Proc. 8th Latin-American Theoretical Informatics Symposium (LATIN'06)*, Springer Verlag LNCS #3887, pp. 502-514, 2006.
- [17] Guerraoui R. and Raynal M., The Alpha and Omega of Asynchronous Consensus. *Tech Report #1676*, IRISA, Université de Rennes (France), 2005. <http://www.irisa.fr/bibli/publi/pi/2005/1676/1676.html>, (Submitted).
- [18] Herlihy M.P., Wait-Free Synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149, 1991.
- [19] Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923,, 1999.
- [20] Lamport L., The Part-Time Parliament. *ACM Transactions on Computer Systems*, 16(2):133-169; 1998.
- [21] Moir M. and Anderson J.H., Wait-Free Algorithms for Fast, Long-Lived Renaming. *Science of Computer Programming*, 25:1-39, 1995.
- [22] Neiger G., Failure Detectors and the Wait-Free Hierarchy. *Proc. 14th Int'l ACM Symposium on Principles of Distributed Computing (PODC'95)*, ACM Press, pp. 100-109, Ottawa (Canada), August 1995.
- [23] Saks M. and Zaharoglou F., Wait-Free k -Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.

A Borowsky-Gafni's adaptive $(2h - 1)$ -renaming algorithm

This appendix describes an adaptive renaming algorithm that, given h participating processes in a set of H processes ($h \leq H$), provides these processes with a renaming space whose size is $M = 2h - 1$. As indicated in the paper, several such algorithms have been proposed (e.g., [2, 4, 9]). We present here the algorithm proposed by Borowski and Gafni [9] as it naturally belongs to Gafni's reduction land.

Data structures The algorithm uses a set of ladder objects as defined in Section 4.2. Each ladder provides an operation denoted `participating_set()` that satisfies the self-membership, comparability and immediacy properties defined in Section 3.2. As we have seen, these objects can be wait-free implemented in asynchronous read/write atomic register systems.

Each ladder is identified $LADDER[tag]$ where tag specifies a ladder among several ladders. A tag is a sequence of integers, which means that the tags $(0, 1)$, $(0, 2)$ and $(0, 1, 3)$ are pointers to three different ladders. More generally, the set of ladders has a tree structure, $LADDER[(0)]$ denoting the root ladder object. The operation \oplus is used to define a new tag from a previous tag (line 04). As an example, the tag $(0, 1) \oplus 5$ is the sequence $(0, 1, 5)$. Moreover, $LADDER[(0, 1, 5)]$ is then a child of $LADDER[(0, 1)]$.

Each process p_i manages three local variables: dir_i , $slot_i$ and tag_i ; $dir_i \in \{up, down\}$ (each one being the opposite of the other); $slot_i \in [0..2H - 1]$, and tag_i is a sequence of integers that allows accessing a ladder object. Initially, $dir_i = up$, $slot_i = 0$ and $tag_i = (0)$.

```

operation BG_rename( $tag_i, slot_i, dir_i, id_i$ )
(01)  $S_i \leftarrow LADDER[tag_i].participating\_set(id_i)$ ;
(02) if ( $dir_i = up$ ) then  $slot_i \leftarrow slot_i + (2|S_i| - 1)$  else  $slot_i \leftarrow slot_i - (2|S_i| - 1)$  end_if;
(03) if ( $id_i = \max(\{id \mid id \in S_i\})$ ) then return( $slot_i$ )
(04) else let  $name_i = BG\_rename(tag_i \oplus |S_i|, slot_i, \neg dir_i, id_i)$ ;
(05) return( $name_i$ )
(06) end_if

```

Figure 8: Borowsky-Gafni's renaming algorithm (code for p_i)

Algorithm description A process p_i invokes the operation $BG_rename(tag_i, slot_i, dir_i, id_i)$ described in Figure 8. Starting from the root, p_i recursively descends along the tree of ladder objects until it stops (line 03). When it enters $BG_rename(tag_i, slot_i, dir_i, id_i)$, p_i first invokes $LADDER[tag_i].participating_set(id_i)$ to obtain a set S_i of participating processes satisfying the self-membership, comparability and immediacy properties. Let us notice that this set can include only processes that have invoked the very same ladder object (identified by tag_i).

Considering the recursive invocations issued by p_i , let S_i^1 be the set obtained by p_i during its first invocation, S_i^2 be the set obtained by its second invocation, etc. A process p_i considers smaller and smaller renaming spaces until it obtains its final name. These renaming spaces are defined at line 02. Thanks to the direction parameter dir_i that takes alternate values, we have the following. Let $L_i^1 = 2|S_i^1| - 1$.

The first renaming space is $rs_i^1 = [1..L_i^1]$ (notice that $L_i^1 \leq 2h - 1$, where h is the number of participating processes). Let $L_i^2 = 2|S_i^2| - 1$. The second renaming space used by p_i (if needed) is $rs_i^2 = [L_i^1 - L_i^2..L_i^1]$. Similarly, let $L_i^3 = 2|S_i^3| - 1$. The third renaming space used by p_i is then $rs_i^3 = [L_i^1 - L_i^2..L_i^1 - L_i^2 + L_i^3]$; etc. We have $rs_i^{x+1} \subseteq rs_i^x$. The process p_i stops descending the ladder tree when, during its x th recursive call, it obtains a set S_i^x such that id_i is the greatest identity in that set (line 03). Let us observe that, when we consider a given depth x of the ladder tree, there is at least one process p_c such that $id_c = \max(\{id \mid id \in S_c^x\})$, from which it follows that each process terminates the algorithm after at most h recursive calls. It is easy to see that the final renaming space that the processes can occupy is $[1..2h - 1]$.

Let $tag[1]^x, tag[2]^x, \dots, tag[z]^x$ be the set of different tags used at the depth x of the ladder tree. The algorithm ensures the following property (from which follows the fact that no two of the $h \leq H$ processes obtain the same name). If $\alpha \neq \beta$, the renaming spaces obtained by a process p_i and a process p_j that invoke $LADDER[tag[\alpha]^x].participating_set(id_i)$ and $LADDER[tag[\beta]^x].participating_set(id_j)$, respectively, have an empty intersection. For more details on this very elegant wait-free algorithm, the reader can consult [9].

In search of the holy grail: Looking for the weakest failure detector for wait-free set agreement

Michel Raynal and Corentin Travers

IRISA, Université de Rennes 1, Campus de Beaulieu, 35042 Rennes, France
{raynal|ctravers}@irisa.fr

Abstract. Asynchronous failure detector-based set agreement algorithms proposed so far assume that all the processes participate in the algorithm. This means that (at least) the processes that do not crash propose a value and consequently execute the algorithm. It follows that these algorithms can block forever (preventing the correct processes from terminating) when there are correct processes that do not participate in the algorithm. This paper investigates the wait-free set agreement problem, i.e., the case where the correct participating processes have to decide a value whatever the behavior of the other processes (i.e., the processes that crash and the processes that are correct but do not participate in the algorithm). The paper presents a wait-free set agreement algorithm. This algorithm is based on a leader failure detector class that takes into account the notion of participating processes. Interestingly, this algorithm enjoys a first class property, namely, design simplicity.

Keywords: Asynchronous algorithm, Asynchronous system, Atomic register, Consensus, Leader oracle, Participating process, Set agreement, Shared object, Wait-free algorithm.

1 Introduction

The consensus problem Consensus is a fundamental fault-tolerant distributed computing problem. As soon as processes cooperate, they have to agree in one way or another. This is exactly what the consensus problem captures: it allows a set of processes to agree on a critical data (called value, decision, state, etc.). Consensus can be informally defined as follows. Each process proposes a value, and a process that is not faulty has to decide a value (termination), such that there is a single decided value (agreement)¹, and that value is a proposed value (validity).

It is well-known that the consensus problem cannot be solved in asynchronous systems prone to even a single process crash, be these systems read/write shared memory systems [13], or message-passing systems [5]. So, one way to circumvent this impossibility is to enrich the asynchronous system with additional objects that are strong enough to allow solving consensus.

¹ We consider here the uniform version of the consensus problem. A faulty process that decides has to decide the same value as the non-faulty processes.

Shared memory systems equipped with objects defined with a sequential specification and more powerful than traditional atomic read/write registers have been investigated. This line of research has produced the notion of *consensus number* that can be associated with each object type defined by a sequential specification [9]. The consensus number of a type is the maximum number of processes for which objects of that type (together with atomic registers) allows solving consensus. For example, the objects provided with a `Test&Set()` operation have consensus number 2, while the objects provided with a `Compare&Swap()` operation have consensus number $+\infty$. The consensus number notion has allowed to establish a hierarchy among the objects (with a sequential specification) according to the synchronization power of the operations they provide to the processes [9].

Another research direction has been the investigation of objects that provide processes with information on failures, namely, the objects called *failure detectors* [2]. A failure detector class² is defined by abstract properties that state which information on failure is provided to the processes. According to the quality of that information, several classes can be defined. Differently from an atomic register or a `Compare&Swap` object, a failure detector has no sequential specification.

As far as one is interested in solving the consensus problem in an asynchronous system prone to process crashes, it has been shown that Ω is the weakest failure detector class that allows solving consensus in such a context [3]. “Weakest” means that any failure detector that allows solving consensus provides information on failures that allows building a failure detector of the class Ω .

A failure detector of the class Ω provides the processes with a primitive, denoted `leader()`, that returns a process identity each time it is called, and eventually always returns the same identity that is the id of a correct process, i.e., a process that does not crash when we consider crash failures. (Examples of message-passing Ω -based consensus algorithms can be found in [7, 12, 16]. These algorithms assume a majority of correct processes, which is a necessary requirement for Ω -based message-passing consensus algorithms.)

The set agreement problem The k -set agreement problem [4] generalizes the consensus problem: it weakens the constraint on the number of decided values by permitting up to k different values to be decided (consensus is 1-set agreement). While k -set agreement can be easily solved in asynchronous systems where the number t of processes that crash is $< k$ (each of a set of k predetermined processes broadcasts its value, and a process decides the first value it receives), this problem has no solution when $k \geq t$ [1, 11, 19]. The failure detector approach to solve the k -set agreement problem in message-passing systems has been investigated in [10, 14, 15, 20].

While (as indicated before) it has been established that Ω is the weakest failure detector class for solving consensus [3], let us remind that finding the weakest failure detector class for solving k -set agreement for $k > 1$ is still an open problem.

² We employ the words “failure detector class” instead of “failure detector type”, as it is the word traditionally used in the literature devoted to failure detectors.

The main question Failure detector-based consensus algorithms implicitly consider that all the processes participate in the consensus algorithm, namely, any process that does not crash is implicitly assumed to propose a value and execute the algorithm. This is also an implicit assumption in the statement that Ω is the weakest failure detector to solve the consensus problem [3]. Basically, an Ω -based consensus algorithm uses the eventual leader to eventually impose the same value to all the processes. As the algorithm does not know when the leader is elected, its main work consists in guaranteeing that no two different values can be decided before the eventual leader is elected. The algorithm uses the eventual leader to *help* decide all the processes that do not crash.

The previous observation raises the following question: What does happen if the process that is the eventual leader does not participate in the consensus algorithm? It appears that the algorithm can then block forever, and consequently the termination property can no longer be guaranteed.

So, a fundamental question is the following: *What is the weakest failure detector to solve the consensus (or, more generally, the k -set agreement) problem when only a subset of the correct processes (not known a priori) propose a value and participate in the agreement problem?* This question can be reformulated as follows: What are the weakest failure detector classes to *wait-free* solve the consensus and the k -set agreement problems? Wait-free means here that a process that proposes a value and does not crash has to decide, whatever the behavior of the other processes (they can participate or not, and be correct or not). The previous observation on Ω shows that a failure detector of that class cannot be the weakest to wait-free solve the consensus problem.

Content of the paper Answering the previous question requires to investigate new failure detector classes and show that one of them allows solving k -set agreement (sufficiency part) while being the weakest (necessity part). This paper addresses the sufficiency part. (On the necessity side, although we don't have yet formal results, we currently are inclined towards thinking that the failure detector class Ω_*^k -see below- is the weakest failure detector class for wait-free solving k -set agreement.)

More precisely, the paper presents a failure detector-based algorithm for shared memory systems that wait-free solves the k -set agreement problem whatever the number p of participating processes, and their behavior, in a set of n processes³. This algorithm assumes that, in addition to single-writer/multi-readers atomic registers, the shared memory provides the processes with a failure detector object of a class that we denote Ω_*^k . That class is an extension of the failure detector class introduced in [18], and the failure detector classes recently introduced in [6] and [17].

The failure detector class Ω^k introduced in [18] extends the classical Ω class [3] by allowing a set of up to k leaders to be returned by each invocation of the `leader()` primitive (Ω^1 boils down to Ω). The aim of the class Ω^* introduced in [6] is to boost

³ Let us remind that all the algorithms based on an object O with consensus number n allows solving consensus whatever the number ($p \leq n$) and the behavior of the participating processes, i.e., they are wait-free consensus algorithms. (Such an object O has always a sequential specification.) In some sense, this paper looks for a failure detector class that, while being as weak as possible, is as strong as the object O , i.e., a class that allows designing wait-free failure detector-based set agreement algorithms. (Failure detectors cannot be defined from a sequential specification.)

obstruction-free algorithms into non-blocking algorithms. That paper also shows that this failure detector class is the weakest for such a boosting. The failure detector class introduced in [17] extends Ω^k by explicitly referring to the notion of participating processes. It has been introduced to circumvent the $2p - 1$ lower bound of the renaming problem [11] (where p is the number of participating processes). Using such a failure detector, the proposed renaming algorithm provides the processes with a renaming space whose size is reduced from $2p - 1$ to $2p - \lceil \frac{p}{k} \rceil$ (where the value k comes from “ k ”-set agreement).

Roadmap The paper is made up of 6 sections. Section 2 presents the computation model. Section 3 introduces the failure detector class Ω_*^k . Then, Section 4 presents the Ω_*^k -based k -set algorithm. This algorithm uses an underlying object denoted KA . So, Section 5 presents an algorithm constructing a KA object from atomic read/write registers. Finally, Section 6 concludes the paper.

2 Asynchronous system model

2.1 Process and communication model

Process model The system consists of n sequential processes that we denote p_1, \dots, p_n . A process can crash. Given an execution, a process that crashes is said to be *faulty*, otherwise it is *correct* in that execution. Each process progresses at its own speed, which means that the system is asynchronous. In the following, *Correct* denotes the set of processes that are correct in the run that is considered.

Coordination model The processes cooperate and communicate through two types of reliable objects: two arrays of single-writer/multi-reader atomic registers and a shared object that we call KA (as shown in Section 5, such an object can be built from single-writer/multi-reader atomic registers). The processes are also provided with a failure detector object of the class Ω_*^k (see below).

Identifiers with upper case letters are used to denote shared objects. Lower case letters are used to denote local variables; in that case the process index appears as a subscript. As an example, $part_i[j]$ denotes the j th entry of a local array of the process p_i , while $PART[j]$ denotes the j th entry of the shared array $PART$.

2.2 The KA object

The KA object is a variant of a round-based object introduced in [8] to capture the safety properties of Paxos-like consensus algorithms [8, 12]. This object provides the processes with an operation denoted $\text{alpha_propose}_k()$. That operation has two input parameters: the value (v_i) proposed by the invoking process p_i , and a round number (r_i). The round numbers play the role of a logical time and allows identifying the invocations. The KA object assumes that no two processes use the same round numbers, and successive invocations by the same process use increasing round numbers. Given a KA object, the invocations $\text{alpha_propose}_k()$ satisfy the following properties (\perp is a default value that cannot be proposed by a process):

- Termination (wait-free): an invocation of $\text{alpha_propose}_k()$ by a correct process always terminates (whatever the behavior of the other processes).
- Validity: the value returned by any invocation $\text{alpha_propose}_k()$ is a proposed value or \perp .
- Agreement: At most k different non- \perp values can be returned by the whole set of $\text{alpha_propose}_k()$ invocations.
- Convergence: If there is a time after which the operation $\text{alpha_propose}_k()$ is invoked infinitely often, and these invocations are issued by an (unknown but fixed) set of at most k processes, there is then a time after which none of these invocations returns \perp .

A KA object can store up to k non- \perp different values. A process invokes it with a value to store and obtains a value in return. If it is permanently accessed concurrently by more than k processes, the KA object might store anything. If there is a period during which it is accessed concurrently by at most $k' \leq k$ processes, it stores forever the corresponding k' proposed values.

3 The failure detector class Ω_*^k

3.1 Definition

A failure detector of the class Ω_*^k provides the processes with an operation denoted $\text{leader}()$. (As indicated in the introduction, this definition is inspired by the leader failure detector classes introduced in [6, 17, 18].) When a process p_i invokes that operation, it provides it with an input parameter, namely a set X of processes, and obtains a set of process identities as a result⁴.

The semantics of Ω_*^k is based on a notion of time, whose domain is the set of integers. It is important to notice that this notion of time is not accessible to the processes. An invocation of $\text{leader}(X)$ by a process p_i is *meaningful* if $i \in X$. If $i \notin X$, it is *meaningless*. The primitive $\text{leader}()$ is defined by the following properties:

- Termination (wait-free): Any invocation of $\text{leader}()$ by a correct process always terminates (whatever the behavior of the other processes).
- Triviality: A meaningless invocation returns any set of processes.
- Eventual multi-leadership for each input set X : For any $X \subseteq \Pi$, such that $X \cap \text{Correct} \neq \emptyset$, there is a time τ_X such that, $\forall \tau \geq \tau_X$, all the meaningful $\text{leader}(X)$ invocations (that terminate) return the same set L_X and this set is such that:
 - $|L_X| \leq k$.
 - $L_X \cap X \cap \text{Correct} \neq \emptyset$.

⁴ The definition of Ω_*^k is not expressed in the framework introduced by Chandra and Toueg to define failure detector classes [2]. More precisely, in their framework, the failure detector operation that a process can issue has no input parameter. It would be possible to express Ω_*^k in their framework. We don't do it to have simpler statements and make the presentation easier to understand.

The intuition that underlies this definition is the following. The set X passed as input parameter by the invoking process p_i is the set of all the processes that p_i considers as being currently *participating* in the computation. (This also motivates the notion of meaningful and meaningless invocations: an invoking process is trivially participating).

Given a set X of participating processes that invoke $\text{leader}(X)$, the eventual multi-leadership property states that there is a time after which these processes obtain the same set L_X of at most k leaders, and at least one of them is a correct process of X . Let us observe that the (at most $k - 1$) other processes of L_X can be any subset of processes (correct or not, participating or not).

It is important to notice that the time τ_X from which this property occurs is not known by the processes. Moreover, before that time, there is an anarchy period during which each process, as far as its $\text{leader}(X)$ invocations are concerned, can obtain different sets of any number of leaders. Let us also observe that if a process p_i issues two meaningful invocations $\text{leader}(X1)$ and $\text{leader}(X2)$ with $X1 \neq X2$, there is no relation linking L_{X1} and L_{X2} , whatever the values of $X1$ and $X2$ (e.g., the fact that $X1 \subset X2$ imposes no particular constraint involving L_{X1} and L_{X2}).

Let us consider an execution in which all the invocations $\text{leader}(X)$ are such that $X = \Pi$ (the whole set of processes are always considered as participating). In that case, Ω_*^k boils down to the failure detector class denoted Ω^k introduced in [18]. If additionally, $k = 1$, we obtain the classical leader failure detector Ω introduced in [3]. When $k = 1$, Ω_*^k boils down to the failure detector class introduced in [6]. It is shown in [6] that Ω is weaker than Ω_*^1 that in turn is weaker than $\diamond\mathcal{P}$ (the class of eventually perfect failure detectors: after some finite but unknown time, these failure detectors suspect all the crashed processes and only them [2]).

3.2 The family $\{\Omega_*^k\}_{1 \leq k \leq n}$

It follows from the definition of Ω_*^k , that the failure detector class family $\{\Omega_*^k\}_{1 \leq k \leq n}$ is such that $\Omega_*^k \subset \Omega_*^{k+1}$.

Moreover, as just indicated, when all the $\text{leader}(X)$ invocations are such that $X = \Pi$, Ω_*^k boils down to Ω^k (as defined in [18]), from which it follows that we have $\Omega^k \subseteq \Omega_*^k$. More generally, the failure detector classes Ω^k and Ω_*^k are related as indicated in Figure 1 where a plain arrow means “includes”, while a dotted arrow means “does not include”.

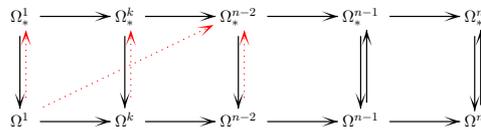


Fig. 1. Wait-free (ir)reducibility results between the families $(\Omega_*^x)_{1 \leq x \leq n}$ and $(\Omega^y)_{1 \leq y \leq n}$

The facts (1) $\Omega^k \subseteq \Omega_*^k$, and (2) Ω^n and Ω_*^n are the same class, follows directly from their definitions. It is important to notice that, $\forall k : 1 \leq k < n - 1, \forall k' : 1 \leq k' \leq n$, it is not possible to build a failure detector of Ω_*^k from a failure detector of class $\Omega^{k'}$. Differently, for $k = n - 1$, the following very simple algorithm (where $leader()$ is the primitive provided by Ω^{n-1}) builds a failure detector of Ω_*^{n-1} from any failure detector of Ω^{n-1} :

operation $leader(X)$: **if** $X = \perp$ **then** $return (leader())$ **else** $return (X)$ **end_if.**

4 Ω_*^k -based k -set agreement

4.1 Wait-free k -set agreement

The k -set agreement has been informally stated in the introduction. It has been defined in [4]. The parameter k of the set agreement can be seen as the degree of coordination associated with the corresponding instance of the problem. The smaller k , the more coordination among the processes: $k = 1$ means the strongest possible coordination (this is the consensus problem), while $k = n$ means no coordination at all. More precisely, in a set of n processes, each of a subset of $p \geq 1$ processes proposes a value. These processes are the *participating* processes. The wait-free k -set agreement is defined by the following properties:

- Termination (wait-free): a correct process that proposes a value decides a value (whatever the behavior of the other processes).
- Agreement: no more than k different values are decided.
- Validity: a decided value is a proposed value.

4.2 Principles and description of the algorithm

The k -set agreement algorithm is described in Figure 2. A process p_i that participates in the k -set agreement invokes the operation $kset_propose_k(v_i)$ where v_i is the value it proposes. If it does not crash, it terminates that operation when it executes the statement $return(decided_i)$ (line 11) where $decided_i$ is the value it decides.

Shared objects The processes share three objects:

- A KA object. A process p_i accesses it by invoking $KA.alpha_propose_k(r_i, v_i)$ where r_i is a round number and the value v_i proposed by p_i (line 07). Due to the properties of the KA object, the value returned by such an invocation is a proposed value or \perp .
- An array of atomic single-writer/multi-reader boolean registers, $PART[1..n]$. The register $PART[i]$ can be updated only by p_i ; it can read by all the processes. Each entry $PART[i]$ is initialized to *false*. $PART[i]$ is switched to *true* to indicate that p_i is now participating in the k -set agreement (line 01). $PART[i]$ is updated at most once.

- An array of atomic single-writer/multi-reader registers denoted $DEC[1..n]$. $DEC[i]$ can be written only by p_i . It can read by all the processes. Each entry $DEC[i]$ is initialized to \perp (a value that cannot be proposed by the processes). When it is updated to a non- \perp value v , that value v can be decided by any process. It is updated (to such a value v or \perp) each time p_i invokes $KA.alpha_propose_k(r_i, v_i)$ to store the value returned by that invocation (line 07).

The algorithm The behavior of a process is pretty simple. As in Paxos, it decouples the safety part from the wait-free/termination part. The safety is ensured thanks to the KA object, while the liveness rests on Ω_*^k .

After it has registered its participation (line 01), a process p_i executes a **while** loop (lines 03-09) until it finds a non- \perp entry in the $DEC[1..n]$ array. When this occurs, p_i decides such a value (lines 03 and 10-11).

Each time it executes the **while** loop, p_i first computes its local view (denoted $part_i$) of the set of the processes it perceives as being the participating processes (line 04). It then uses this participating set to invoke $Leader_i()$ (line 05). If it does not belong to the set returned by $Leader_i(part_i)$, p_i continues looping. Otherwise (it then belongs to set of leaders), p_i invokes the KA object (line 07) to try to obtain a non- \perp value from that object. The local variable r_i is used by p_i to define the round number it uses when it invokes the KA object. It is easy to see from the management of r_i at line 02 and line 06 that each process uses increasing round numbers, and that no two processes use the same round numbers (a necessary requirement for using the KA object). The properties of KA ensure that no more than k values are decided, while the properties of Ω_*^k ensure that all the correct participating processes do terminate.

```

operation kset_proposek(vi):
(1)  PART[i] ← true;
(2)  ri ← (i - n);
(3)  while (∀j : DEC[j] = ⊥) do
(4)    parti ← {j : PART[j] ≠ ⊥};
(5)    leadersi ← Leaderi(parti);
(6)    if (i ∈ leadersi) then ri ← ri + n;
(7)                                     DEC[i] ← KA.alpha_proposek(ri, vi)
(8)    end_if
(9)  end_while;
(10) let decidedi = any DEC[j] ≠ ⊥;
(11) return(decidedi)

```

Fig. 2. An Ω_*^k -based k -set agreement algorithm (code for p_i)

4.3 Proof of the algorithm

Theorem 1. *The algorithm described in Figure 2 wait-free solves the k -set agreement problem whatever the number p of participating processes in a set of n processes (this number p being a priori unknown).*

Proof

Validity The validity property follows from the following observations:

- The value \perp cannot be decided (lines 03 and 10).
- The $DEC[1..n]$ array can contain only \perp or values that have been proposed to the KA object (line 07).
- Any value v_i proposed to the KA object is a value proposed to the k -set agreement.

Agreement The agreement property follows directly from the agreement property of the KA object (that states that at most k non- \perp values can be returned from that object).

Termination (wait-free) If an entry of $DEC[1..n]$ is eventually set to a non- \perp value, it follows from the test of line 03 that any correct participating process terminates. So, let us assume by contradiction that no entry of $DEC[1..n]$ is ever set to a non- \perp value. Let us first observe that all the $leader()$ invocations issued by the processes are meaningful.

If no correct participating process decides, there is a time τ_0 after which we have the following:

- All the participating processes have entered the algorithm, and consequently the array $PART[1..n]$ determines the whole set of participating processes. Let X be this set of processes.
- all the $leader()$ invocations have X as input parameter.

It follows from the eventual multi-leadership property associated with X , that there is a time $\tau_X \geq \tau_0$ such that, for all the times $\tau \geq \tau_X$, all the invocations of $leader(X)$ return the same set L_X of at most k processes, and this set includes at least one correct participating process.

As no process decides (assumption) and each $\alpha_propose_k()$ invocation issued by a correct process returns (termination property of the KA object), the correct participating processes of the set X execute $KA.\alpha_propose_k()$ infinitely often (lines 06-07). It then follows from the convergence property of the KA object that these processes obtain non- \perp values, and deposit these values in the array $DEC[1..n]$. A contradiction.

□*Theorem 1*

5 Building a KA object from registers

This section presents an implementation of a KA object from single-writer/multi-readers atomic registers. As already indicated, this algorithm is inspired from Paxos-like algorithms [8, 12].

5.1 Implementing KA

An algorithm constructing a KA object is described in Figure 3. It uses an array of single-writer/multi-reader atomic registers REG . As previously, $REG[i]$ can be written only by p_i . A register $REG[i]$ is made up of three fields $REG[i].lre$, $REG[i].lrww$ and $REG[i].val$ whose meaning is the following ($REG[i]$ is initialized to $\langle 0, 0, \perp \rangle$):

- $REG[i].lre$ stores the number of the last round entered by p_i . It can be seen as the logical date of the last invocation issued by p_i .
- $REG[i].lrww$ and $REG[i].val$ constitute a pair of related values: $REG[i].lrww$ stores the number of the last round with a write of a value in the field $REG[i].val$. So, $REG[i].lrww$ is the logical date of the last write in $REG[i].val$.

(To simplify the writing of the algorithm, we consider that each field of a register can be written separately. This poses no problem as each register is single writer. A writer can consequently keep a copy of the last value it has written in each register field and rewrite it when that value is not modified.)

```

operation alpha_proposek( $r, v$ ):
(1)   $REG[i].lre \leftarrow r$ ;
(2)  for  $j \in \{1, \dots, n\}$  do  $reg_i[j] \leftarrow REG[j]$  end_do;
(3)  let  $value_i$  be  $reg_i[j].val$  where  $j$  is such that  $\forall x : reg_i[j].lrww \geq reg_i[x].lrww$ ;
(4)  if ( $value_i = \perp$ ) then  $value_i \leftarrow v$  end_if;
(5)   $REG[i].(lrww, v) \leftarrow (r, value_i)$ ;
(6)  for  $j \in \{1, \dots, n\}$  do  $reg_i[j] \leftarrow REG[j]$  end_do;
(7)  if ( $|\{j | reg_i[j].lre \geq r\}| > k$ ) then return( $\perp$ )
(8)  else return( $value_i$ ) end_if

```

Fig. 3. A KA object algorithm (code for p_i)

The principle that underlies the algorithm is very simple: it consists in using a logical time frame (represented here by the round numbers) to timestamp the invocations, and answering \perp when the timestamp of the corresponding invocation does not lie within the k highest dates (registered in $REG[1..n].lre$). To that end, the algorithm proceeds as follows:

- Step 1 (lines 01-02): Access the shared registers.
 - When a process p_i invokes $\text{alpha_propose}_k(r, v)$, it first informs the other processes that the KA object has attained (at least) the date r (line 01). Then p_i reads all the registers in any order (line 02) to know the last values (if any) written by the other processes.
- Step 2 (lines 03-05): Determination and writing of a value.
 - Then, p_i determines a value. In order not to violate the agreement property, it selects the last value (“last” according to the round numbers/logical dates) that has been deposited in a register $REG[j]$. If there is no such value it considers its own value v . After this determination, p_i writes in $REG[i]$ the value it has determined, together with its round number (line 05).

- Step 3 (lines 06-08): Commit or abort.
 - p_i reads again the shared registers to know the progress of the other processes (measured by their round numbers), line 07. If it discovers it is “late”, p_i aborts returning \perp . (Let us observe that this preserves the agreement property.) “To be late” means that the current date r of p_i does not lie within the window defined by the k highest dates (round numbers) currently entered by the processes (these round numbers/dates are registered in the field lre of each entry of the array $REG[1..n]$).
 - Otherwise, p_i is not late. It then returns (“commits”) $value_i$ (line 08). Let us observe that, as the notion of “being late” is defined with respect to a window of k dates (round numbers), it is possible that up to k processes are not late and return concurrently up to k distinct non- \perp values.

It directly follows from the code that the algorithm is wait-free. Moreover, in order to expedite the $\alpha_propose_k()$ operation, it is possible to insert the statement

if ($|\{j | reg_i[j].lre \geq r\}| > k$) **then** return(\perp) **end_if**

between the line 02 and the line 03. This allows the invoking process to return \perp when, just after entering the $\alpha_propose_k()$ operation, it discovers it is late.

5.2 Proof of the KA object

Theorem 2. *The algorithm described in Figure 3 wait-free implements a KA object.*

Proof

Termination (wait-free) This property follows directly from the code of the algorithm (the only loops are **for** loops that trivially terminate when the invoking process is correct).

Validity Let us observe that if a value v is written in $REG[i].val$, that value has been previously passed as a parameter in an $\alpha_propose_k()$ invocation. The validity property follows from this observation and the fact that only \perp or a value written in a register $REG[i]$ can be returned from an $\alpha_propose_k()$ invocation.

Convergence Let τ be a time after which there is a set of $k' \leq k$ processes such that each of them invokes $\alpha_propose_k()$ infinitely often. This means that, from τ , the values of $n - k'$ registers $REG[x]$ are no longer modified. Consequently, as the k' processes p_j repeatedly invoke $\alpha_propose_k()$, there is a time $\tau' \geq \tau$ after which each $REG[j].lre$ becomes greater than any $REG[x].lre$ that is no longer modified. There is consequently a time $\tau'' \geq \tau'$ after which the k' processes are such that their registers $REG[j].lre$ contain forever the k greatest timestamp values. It follows from the test done at line 07 that, after τ'' , no $\alpha_propose_k()$ invocation by one of these k' processes can be aborted. Consequently, each of them returns a non- \perp value at line 08.

Agreement If all invocations returns \perp , the agreement property is trivially satisfied. So, let us consider an execution in which at least one $\alpha_propose_k()$ invocation returns a non- \perp value. To prove the agreement property we show that:

- Before the first non- \perp value is returned by an invocation, there is a time at which the algorithm has determined a set V of at least one and at most k non- \perp values⁵.
- Any value $v \neq \perp$ returned by an invocation is a value of V .

To simplify the reasoning, and without loss of generality, we assume that a process that repeatedly invokes $\text{alpha_propose}_k()$, stops invoking that operation as soon as it returns a non- \perp value at line 08.

1. Invariants. $\forall j \in \{1, \dots, n\}$:
 - $REG[j].lre$ is increasing (assumption on the successive round numbers used by p_j).
 - $REG[j].lrw \leq REG[j].lre$ (because p_j executes line 05 after line 01).
2. Among all the invocations that execute the test of line 07, let \mathcal{I} be the subset of invocations for which the predicate $|\{j | reg_i[j].lre \geq r\}| \leq k$ is true. (This means that any invocation of \mathcal{I} either returns a non- \perp value -at line 08-, or crashes after it has evaluated the predicate at line 07, and before it executes line 08.) Among the invocations of \mathcal{I} , let I be the invocation with the smallest round number. Let p_{j_1} be the process that invoked I and r the corresponding round number.
3. Time instants (see Figure 4).
 - Let τ be the time at which I executes line 05 (statement $REG[j_1] \leftarrow \langle r, r, v \rangle$).
 - Let τ' be the time just after I has finished reading the array $REG[1..n]$. Without loss of generality, we consider that this is the time at which I locally evaluates the predicate of line 07.
 - Let $\tau[j]$ be the time at which I reads $REG[j]$ at line 06. We have $\tau < \tau[j] < \tau'$.

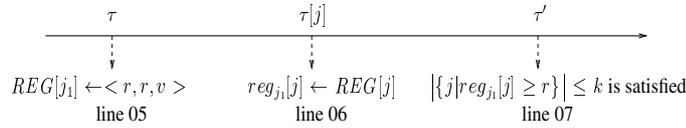


Fig. 4. Time instants with respect to accesses to the registers $REG[1..n]$

4. From $\tau[j] < \tau'$, the fact that predicate $|\{j | reg_{j_1}[j].lre \geq r\}| \leq k$ is true at τ' , and the monotonicity of $REG[j].lre$, we can conclude that a necessary requirement for the predicate $REG[j].lre \geq r$ to be true at τ is that it is true at τ' .
Let $L = \{j_1, \dots, j_x, \dots, j_\ell\}$ be the set of processes p_j such that $REG[j].lre \geq r$ is true at τ . As the predicate $|\{j | reg_i[j].lre \geq r\}| \leq k$ is true at τ' , we have $1 \leq \ell = |L| \leq k$.
5. From the previous item, we conclude that there are at least $n - \ell \geq n - k$ entries j of the array $REG[1..n]$ such that $REG[j].lre < r$ at time τ . Let \bar{L} denote this set of processes (L and \bar{L} define a partition of the whole set of processes).

⁵ According to the terminology introduced in [2], the set V defines the values that are *locked*. This means that from now on the set of non- \perp values that can be returned is fixed forever: no value outside V can ever be returned.

6. Let the τ -time invocation of p_j be the invocation issued by p_j whose round number is the value of $REG[j].lre$ at time τ (assuming a fictitious initial invocation if needed).
7. The τ -time invocations of the processes p_j in L define a set, denoted V , including at most $\ell \leq k$ values, such that these values are written in $REG[1..n]$ with a write timestamp (value of the field $REG[j].lrww$) that is $\geq r$. This claim follows from the following observation.
 - The τ -time invocation by p_{j_1} (namely I) writes a value and the round number r in $REG[j_1]$.
 - Let $p_{j_x} \in L$, $p_{j_x} \neq p_{j_1}$. From the definition of L , it follows that the round number of the τ -time invocation issued by p_{j_x} is $REG[j_x].lre = r' > r$. When it executes that invocation, p_{j_x} atomically executes $REG[j_x] \leftarrow \langle r', r', v' \rangle$ (if it does not crash before executing the line 05).
 - It is possible that, on one side, no process in L crashes before executing line 05, and, on another side, all the values that are written are different. It consequently follows that up to $\ell \leq k$ different values (with a write timestamp $lrww \geq r$) can be written in $REG[1..n]$. Hence, V can contain up to k values.
 - Moreover, it is also possible that each process in L returns at line 08 the value it has selected at line 05 (this depends on the value of the predicate evaluated at line 07). Consequently each value of V can potentially be returned.
8. Given an execution, the previous item has extracted a non-empty set V of at most k non- \perp values that can be returned. We now show that (1) from τ , only values of V can be written in $REG[1..n]$ with a timestamp field ($lrww$) greater than r , and (2) a non- \perp value returned by an invocation is necessarily a value of V .
 - (a) The τ -time invocation issued by any $p_j \in \bar{L}$ has a round number $REG[j].lre$ that is smaller than $REG[j_1].lre = r$ (this follows from the definition of \bar{L}). As by definition, r is the smallest round number during which a process finds true the predicate of line 07, it follows that any $p_j \in \bar{L}$ needs to issue an invocation with a round number greater than r to have a chance to return a non- \perp value.
 - (b) Let \mathcal{I}' be the set of all the invocations that have a round number greater than r . They are issued by the processes of \bar{L} or the processes of L whose τ -time invocation has returned \perp at line 07. Let us observe that any invocation of \mathcal{I}' starts after τ .
 Let I' be the first invocation of \mathcal{I}' that executes 05. I' (issued by some process p_j) selects (at line 03) a value $value_j$ from a register $REG[y]$ such that $REG[y].lrww \geq REG[j_1].lrww = r$. As up to now, only processes of L have written values in $REG[1..n]$ with a write timestamp ($lrww$) $\geq r$, it follows that I' selects a value from V ⁶. Consequently, this invocation does not add a new value to V .
 Let I'' be the invocation of \mathcal{I}' that is the second to execute line 05. The same reasoning (including now I') applies. Etc. It follows from this induction that a

⁶ It is possible that, when I' reads the array $REG[1..n]$ at line 02, not all the values of V have yet been written in that array. The important points are here that (1) at least one value of V has already been written in the array (namely, $REG[j_1].val$ with the timestamp $REG[j_1].lrww = r$), and (2) any register $REG[x]$ that currently contains a value not in V , is such that $REG[x].lrww < r$.

value written at line 05 by an invocation of T' is a value of V , which proves that only values of V can be written in the array $REG[1..n]$ with a write timestamp greater than r .

- (c) Finally, an invocation that returns a value at line 08, returns the value it has written at line 05. Due to the definition of r , its round number r' is $\geq r$. It follows that the non- \perp value that is returned is a value of V . $\square_{\text{Theorem 2}}$

6 Conclusion

Considering asynchronous systems equipped with a failure detector object, this paper focused on the set agreement problem when only a subset of the processes participate, namely, the *wait-free set agreement* problem. Wait-free means here that a correct process has to decide a value, whatever the behavior of the other processes (that can be correct or not and participate or not).

A wait-free failure detector-based k -set agreement algorithm has been presented. Its design principles follows the Paxos approach, decoupling the way the safety and the termination properties are guaranteed. The algorithm safety is based on an object denoted KA that can be built from single-writer/multi-reader atomic registers. The liveness property is based on a leader failure detector class, denoted Ω_*^k , that takes into account the participating processes. The very existence of the algorithm shows that Ω_*^k is sufficient to wait-free solve the k -set agreement problem. Showing that Ω_*^k is also necessary, or defining a class of weaker failure detectors solving the k -set agreement problem, remains one of the greatest research challenges for the fault-tolerant asynchronous computing theory community.

References

1. Borowsky E. and Gafni E., Generalized FLP Impossibility Results for t -Resilient Asynchronous Computations. *Proc. 25th ACM Symposium on Theory of Computation (STOC'93)*, San Diego (CA), pp. 91-100, 1993.
2. Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225-267, 1996.
3. Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *Journal of the ACM*, 43(4):685-722, 1996.
4. Chaudhuri S., More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation*, 105:132-158, 1993.
5. Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.
6. Guerraoui R., Kapalka M. and Kouznetsov P., The Weakest Failure Detectors to Boost Obstruction-Freedom. *Proc. 20th Symposium on Distributed Computing (DISC'06)*, Springer Verlag LNCS #4167, Stockholm (Sweden), 2006.
7. Guerraoui R. and Raynal M., The Information Structure of Indulgent Consensus. *IEEE Transactions on Computers*. 53(4):453-466, 2004.
8. Guerraoui R. and Raynal M., The Alpha of Asynchronous Consensus. *The Computer Journal*. To appear, 2006.

9. Herlihy M.P., Wait-Free Synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124-149, 1991.
10. Herlihy M.P. and Penso L. D., Tight Bounds for k -Set Agreement with Limited Scope Accuracy Failure Detectors. *Distributed Computing*, 18(2): 157-166, 2005.
11. Herlihy M.P. and Shavit N., The Topological Structure of Asynchronous Computability. *Journal of the ACM*, 46(6):858-923,, 1999.
12. Lamport L., The Part-Time Parliament. *ACM Transactions on Computer Systems*, 16(2):133-169; 1998.
13. Loui M.C., Abu-Amara H., Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing research*, JAI Press, 4:163-183, 1987.
14. Mostéfaoui A., Rajsbaum S., Raynal M. and Travers C., Irreducibility and Additivity of Set Agreement-oriented Failure Detector Classes. *Proc. 25th ACM Symposium on Principles of Distributed Computing PODC'06*, ACM Press, Denver (Colorado), 2006.
15. Mostéfaoui A. and Raynal M., k -Set Agreement with Limited Accuracy Failure Detectors. *Proc. 19th ACM Symposium on Principles of Distributed Computing (PODC'00)*, ACM Press, pp. 143-152, 2000.
16. Mostéfaoui A. and Raynal M., Leader-Based Consensus. *Parallel Processing Letters*, 11(1):95-107, 2001.
17. Mostéfaoui A., Raynal M. and Travers C., Exploring Gafni's reduction land: from Ω^k to wait-free adaptive $(2p - \lceil \frac{p}{k} \rceil)$ -renaming via k -set agreement. *Proc. 20th Symposium on Distributed Computing (DISC'06)*, Springer Verlag LNCS #4167, Stockholm (Sweden), 2006.
18. Neiger G., Failure Detectors and the Wait-free Hierarchy. *Proc. 14th ACM Symp. on Principles of Distributed Computing (PODC'95)*, ACM Press, pp. 100-109, 1995.
19. Saks M. and Zaharoglou F., Wait-Free k -Set Agreement is Impossible: The Topology of Public Knowledge. *SIAM Journal on Computing*, 29(5):1449-1483, 2000.
20. Yang J., Neiger G. and Gafni E., Structured Derivations of Consensus Algorithms for Failure Detectors. *Proc. 17th ACM Symp. on Principles of Distributed Computing (PODC'98)*, ACM Press, pp.297-308, 1998.

Prêt à Voter with re-encryption mixes

P Y A Ryan¹ and S A Schneider²

¹University of Newcastle, ²University of Surrey

Abstract. We present a number of enhancements to the voter verifiable election scheme Prêt à Voter [CRS05]. Firstly, we propose a mechanism for the distributed construction by a set of independent clerks of the ballot forms. This construction leads to proto-ballot forms with the candidate list encrypted and ensures that only a collusion of all the clerks could determine the cryptographic seeds or the onion/candidate list association. This eliminates the need to trust a single authority to keep this information secret. Furthermore, it allows the on-demand decryption and printing of the ballot forms, so eliminating chain of custody issues and the chain voting style attacks against encrypted receipt schemes identified in [RP05].

The ballot forms proposed here use ElGamal randomised encryption so enabling the use of re-encryption mixes for the anonymising tabulation phase in place of the decryption mixes. This has a number of advantages over the RSA decryption mixes used previously: tolerance against failure of any of the mix tellers, full mixing of terms over the Z_p^* space and enabling the mixes and audits to be fully independently rerun if necessary.

1 Introduction

The Prêt à Voter scheme, presented in [CRS05], is a cryptographic voting scheme that enables voter-verifiability: at the time of casting their vote, voters are provided with an encrypted receipt. They can then check, via a secure Web Bulletin Board (WBB), that their receipt is accurately included in a robust anonymising mix process. Various checking mechanisms serve to detect any corruption in any phase of this process: encryption of the vote, recording and transmission of the encrypted ballot receipt and the decryptions of the votes. Full details can be found in [CRS05]. Henceforth we will refer to this version of the scheme as Prêt à Voter'05.

Prêt à Voter seeks to achieve the goals of accuracy and ballot secrecy with minimal trust in the system: software, hardware, officials. Assurance is achieved through a high degree of transparency and we thus verify the correctness of the election rather than attempting to verify the system.

This scheme has the benefit of providing a very simple and familiar voter experience, but certain vulnerabilities and trust assumptions have been identified, see [RP05]. In this paper we present a number of enhancements designed to counter these threats and eliminate the need for these trust assumptions.

The construction of the ballot forms presented here also enables the use of re-encryption mixes in the anonymising/tabulation phase. This also provides a number of advantages over the RSA/decryption mixes of Prêt à Voter'05.

The structure of the paper is as follows: in the next section we give the key elements of Prêt à Voter'05. Section 3 summarises some of the threats to and trust assumptions needed in Prêt à Voter'05. Section 4 presents the distributed construction of encrypted ballot forms. Sections 5 and 6 describe how these forms can be used in the vote casting process. Section 7 describes the use of this construction for re-encryption mixes during the anonymising and tabulation phase. Sections 8 and 9 describe the new auditing procedures required for the new ElGamal style ballot forms. Sections 10 and 11 discuss some further extensions to deal with more general voting methods and remote voting.

2 Outline of Prêt à Voter 2005

We now present an overview of the Prêt à Voter voter-verifiable scheme. Voters select at random a ballot form, an example of which is shown in Figure 1.

Obelix	
Asterix	
Panoramix	
Idefix	
	<i>7rJ94K</i>

Fig. 1. Prêt à Voter ballot form

In the booth, the voter makes her selection in the usual way by placing a cross in the right hand column against the candidate of choice, or, in the case of a Single Transferable Vote (STV) system for example, they mark their ranking against the candidates. Once the selection has been marked, the left hand strip is detached and discarded. The remaining right hand strip now constitutes the receipt, as shown in Figure 2.

X
<i>7rJ94K</i>

Fig. 2. Prêt à Voter ballot receipt

The voter now exits the booth and casts their vote in the presence of an official. The ballot receipt is placed under an optical reader or similar device that records the random value at the bottom of the strip and an index value indicating the cell into which the X was marked. The receipt is digitally signed and franked and the voter now retains this as their receipt.

Possession of a receipt might appear to open up the possibility of coercion or vote-buying. However, the candidate lists on the ballot forms are independently randomised for each ballot form. Thus, with the left hand strip removed, the right hand strip alone does not indicate which way the vote was cast.

The cryptographic value printed on the bottom of the receipt, the ‘onion’, is the key to extraction of the vote. Buried cryptographically in this value is the information needed to reconstruct the candidate list shown on the left hand strip. This information is encrypted under the secret keys shared by a number of tellers. Thus, only the tellers acting in concert are able to reconstruct the candidate order and so interpret the vote value encoded on the receipt.

Once the election has closed, all the receipts are transmitted to a central tabulation server which posts them to a secure WBB. This is an append-only, publicly visible facility. Only the tabulation server, and later the tellers, can write to this and, once written, anything posted to it will remain unchanged. Voters can visit this WBB and confirm that their receipt appears correctly.

After a suitable period, the tellers take over and perform a robust, anonymising, decryption mix on the batch of posted receipts. Various approaches can be used to ensure that the tellers perform the decryptions correctly. Details of this can be found in [CRS05].

Prêt à Voter’05 proposes an Authority responsible for the generation of the entropy for the crypto seeds and prior printing of the ballot forms. Random auditing, by independent organisations, of the forms before, during and after the election serve to detect any attempt by the the Authority to pass off incorrectly formed ballot forms. Later in this paper we propose an alternative approach using on-demand creation and printing of forms and post-auditing.

The Prêt à Voter’05 approach has the advantage of simplicity and results in a very simple and familiar experience for the voters: they simply register, collect a form, mark their selection in the booth and then cast the form.

For full details of the mechanisms used in the 2005 version of the scheme to detect any malfunction or misbehaviour by the devices or processes that comprise the scheme, see [CRS05]. The construction of the ballot forms used here calls for rather different monitoring and auditing mechanisms that we detail later.

3 Threats and trust models

The simplicity of the original scheme, in particular the use of a single authority and the pre-printing and pre-auditing of the ballot forms, comes at a certain

cost: various trust assumptions need to be made. In this section we briefly recall the threats and assumptions of Prêt à Voter'05 identified in [RP05].

3.1 The need to trust the Authority for confidentiality

In Prêt à Voter 2005, a single entity creates the ballot forms. Whilst it is not necessary to trust this entity from the point of view of accuracy, it is necessary to trust it not to leak the ballot form information. Clearly, if the Authority were to leak this information, the scheme would become susceptible to coercion or vote buying.

3.2 Chain of custody

Just as we need to trust the Authority not to leak ballot form information, we also need to assume that mechanisms are in place to ensure that none of this information is leaked during storage and distribution. Various counter-measures are possible: for example, ballot forms could be kept in sealed envelopes to be revealed only by the voters in the booth. Alternatively, a scratch card style mechanism along the lines suggested in [RP05] could be used to conceal the onion value until the voter reveals it at the time of vote casting. The ballot forms would also need to be stored and distributed in locked, sealed boxes. All of these counter-measures are rather procedural in nature and so require various trust assumptions.

3.3 Chain voting

Conventional, pen and paper elections may be vulnerable to a style form of vote buying known as chain voting. The UK system in particular is vulnerable. Here, the ballot forms are a controlled resource: on entering the polling station, the voter is registered and marked off on the electoral roll. They are given a ballot form which they take to the booth, mark and then cast in the ballot box. In principle, officials should observe the voters casting their form.

The attack works as follows: the coercer smuggles a blank ballot form out of the polling station. The controls on the distribution of the forms should make this a little tricky, but in practise there are many ways it could be achieved. Having marked the form for the candidate of their choice, the coercer intercepts a voter as they enter the polling station. The voter is told that if, when they exit the polling station, they hand a fresh, blank form back to the coercer they will receive an reward. The attack can now proceed inductively until a voter decides to cry foul. Note that, once initialised, the controls on the ballot forms works in the coercer's favour: if the voter emerges from the polling station with a blank form, it is a strong indication that they did indeed cast the marked form they were given by the coercer.

3.4 Kleptographic channels

A further, rather subtle vulnerability can occur where a single entity is responsible for creating cryptographic variables: kleptographic attacks as described in [YY96]. The possible relevance of such attacks to cryptographic voting schemes is described in [M. 06]. The idea is that the entity may carefully choose the values of the crypto variables in order to leak information to a colluding party.

In the case of Prêt à Voter, the Authority might choose the seed values in such a way that an agreed, keyed cryptographic hash of the onion value indicates the candidate order. Clearly this may require quite a bit of searching and computation to find suitable values. Note however that such an attack could pass unnoticed: the distribution of seed values would look perfectly random to anyone ignorant of the cryptographic hash function.

4 Distributed generation of encrypted ballot forms

Many of the above attacks stem from the fact that a single entity is able to determine, in the sense of being able both to know and to control, the seed values. We now present a mechanism for the distributed generation of the seed values and ballot forms. Throughout, we will use ElGamal encryption rather than RSA as used in Prêt à Voter'05 and we will work in Z_p^* , p a (large) prime.

An analogous construction is possible for the distributed creation of the RSA, layered onions of Prêt à Voter'05. However, as we want to introduce re-encryption mixes at the tabulation stage, we present the construction for ElGamal encryption here. We note also that the term *onion* is a slight misnomer where ElGamal terms are used but we will retain it here for historical reasons.

The ballot forms will be generated by a set of l clerks in such a way that each contributes to the entropy of the crypto seed and this remains encrypted throughout. Consequently the candidate list, which is derived from the seed, remains concealed and all the clerks would have to collude to determine the seeds values.

We assume a set of decryption tellers who hold the key shares for a threshold ElGamal primitive with public key: (p, α, β_T) . These will act much as the tellers of the original scheme and will be responsible for the final decryption stage after the anonymising, re-encryption mix phase. Details of the anonymising and decryption/tabulation phases will be given in section 7.

We also assume a set of Registrars with threshold secret key shares corresponding to the public key: (p, α, β_R) . These public keys are known to the Clerks and are used in the construction of the ballot forms.

An initial clerk C_0 generates a batch of initial seeds s_i^0 . These seeds are drawn randomly from a binomial distribution centred around 0 with standard deviation σ . σ would probably be chosen to be of order n , the number of candidates.

From these, C_0 generates a batch of pairs of "entangled" onions by encrypting each s_i^0 , actually in the form $\gamma^{-s_i^0}$, under the Registrar key and the Teller key:

$$(\{\gamma^{-s_i^0}\}_{PK_R}, \{\gamma^{-s_i^0}\}_{PK_T}).$$

Expressed as ElGamal encryptions these have the form:

$$(\alpha^{x_i^0}, \beta_R^{x_i^0} \cdot \gamma^{-s_i^0}), (\alpha^{y_i^0}, \beta_T^{y_i^0} \cdot \gamma^{-s_i^0})$$

for fresh random values x_i^0, y_i^0 drawn from Z_p^* .

Notice that, for convenience later, we have encrypted the value $\gamma^{-s_i^0}$ for some generator γ of Z_p^* rather than encrypting s_i^0 directly. The reason for this will become apparent shortly.

The remaining $l - 1$ Clerks now perform re-encryption mixes and transformations on this batch of onion pairs. Each Clerk takes the batch of pairs output by the previous Clerk and performs a combined re-encryption along with an injection of fresh entropy into the seed values. For each pair of onions, the same entropy is injected into the seed value of both onions to ensure that these values continue to match for each pair.

More precisely, for each pair of the batch, the j th Clerk C_j generates a new, random values \bar{x}, \bar{y} and \bar{s} and performs the following mix/transformation on each onion pair of the batch:

$$\begin{aligned} & \{(\alpha^{x_i^{j-1}}, \beta_R^{x_i^{j-1}} \cdot \gamma^{-s_i^{j-1}}), (\alpha^{y_i^{j-1}}, \beta_T^{y_i^{j-1}} \cdot \gamma^{-s_i^{j-1}})\} \\ & \quad \downarrow \\ & \{(\alpha^{x_i^{j-1}} \cdot \alpha^{\bar{x}_i^j}, \beta_R^{x_i^{j-1}} \cdot \beta_R^{\bar{x}_i^j} \cdot \gamma^{-s_i^{j-1}} \cdot \gamma^{-\bar{s}_i^j}), (\alpha^{y_i^{j-1}} \cdot \alpha^{\bar{y}_i^j}, \beta_R^{y_i^{j-1}} \cdot \beta_R^{\bar{y}_i^j} \cdot \gamma^{-s_i^{j-1}} \cdot \gamma^{-\bar{s}_i^j})\} \\ & \quad \downarrow \\ & \{(\alpha^{(x_i^{j-1} + \bar{x}_i^j)}, \beta_R^{(x_i^{j-1} + \bar{x}_i^j)} \cdot \gamma^{-(s_i^{j-1} + \bar{s}_i^j)}), (\alpha^{(y_i^{j-1} + \bar{y}_i^j)}, \beta_R^{(y_i^{j-1} + \bar{y}_i^j)} \cdot \gamma^{-(s_i^{j-1} + \bar{s}_i^j)})\} \\ & \quad \downarrow \\ & \{(\alpha^{x_i^j}, \beta_R^{x_i^j} \cdot \gamma^{-s_i^j}), (\alpha^{y_i^j}, \beta_T^{y_i^j} \cdot \gamma^{-s_i^j})\} \end{aligned}$$

where

$$\begin{aligned} x_i^j &= x_i^{j-1} + \bar{x}_i^j \\ y_i^j &= y_i^{j-1} + \bar{y}_i^j \\ s_i^j &= s_i^{j-1} + \bar{s}_i^j \end{aligned}$$

The \bar{x}, \bar{y} denote fresh random values drawn from from Z_p^* generated by the Clerk during the mix. Similarly the \bar{s} values are freshly created random values except that these are again chosen randomly and independently with a binomial distribution mean 0 and standard deviation σ . Having transformed each onion pair in this way, the Clerk C_j then performs a secret shuffle on the batch and outputs the result to the next Clerk, C_{j+1} .

Thus, each Clerk performs a re-encryption mix along with the injection of further entropy into the seed values \bar{s} .

So the final output after $l - 1$ mixes is a batch of pairs of onions of the form: $\{(\alpha^{x_i}, \beta_R^{x_i} \cdot \gamma^{-s_i}), (\alpha^{y_i}, \beta_T^{y_i} \cdot \gamma^{-s_i})\}$ where:

$$x_i = x_i^l, y_i = y_i^l, s_i = s_i^l$$

thus:

$$x_i = \sum_{i=1}^l \bar{x}^i$$

etc.

The final s_i values will have binomial distribution mean 0 and standard deviation $\sigma\sqrt{l}$.

We will refer to the first onion as the “Registrar onion” or “booth onion” and the second onion as the “Teller onion”.

For each pair, assuming correct behaviour of the clerks, the s values in the two onions should match. We’ll discuss mechanisms to detect corruption of the forms later. As the seed values, and hence the candidate orders, remain encrypted, none of clerks knows the seed values and only if they all acted in collusion could they determine the seed values. These “proto-ballot form” can now be stored and distributed in encrypted form, thus avoiding the chain of custody problems mentioned above. The seed values can now be revealed on demand by a threshold set of the Registrars.

5 On-demand creation of ballot forms

The above construction of the proto-ballot forms means that the ballot form material can be stored and distributed in encrypted form. Once registered at the polling station, voters are assigned at random one of these forms:

<i>onion_L</i>	<i>onion_R</i>

The voter proceeds to the booth in which they find a device that reads the left-hand onion. In the simplest case, the secret key to decrypt the left-hand onions could be held in the devices in the booths. Thus, the left hand onion

could be decrypted in the booth, the seed value s revealed and the candidate order π derived as some agreed function of s . If lodging the keys in a single device is considered rather fragile, the left-hand onion could be encrypted under a threshold key held by a number of registrars. The onions could be transmitted to these registrars and a threshold set of these would then decrypt the onions and return the seed to the booth device.

The candidate list can now be printed by the device in the booth to give a standard Prêt à Voter ballot form:

Obelix	
Asterix	
Panoramix	
Idefix	
$onion_L$	$onion_R$

As an additional precaution, the left-hand onion might be separately destroyed.

The point of the paired onions is now clear: we arrange for the booth device to see only the left hand onion and so it will not know the association of the candidate list with the right hand, teller onion that will appear on the receipt. Various mechanisms are possible to ensure that the booth device does not see the right-hand onion. The scratch strip mechanism could be invoked here again for example: the right-hand onion would be covered by a scratch strip that would only be removed at the time of casting, or even at some time after casting. The voter only really needs to reveal the teller onion when they come to check their receipt on the WBB.

Strictly speaking, the l th clerk in collusion with the booth device could form the candidate list/onion association. Elaborations of the scheme to counter the threat of such collusion attacks are the subject of ongoing research.

6 Supervised casting of a ballot

The voter in the booth now has a “conventional” Prêt à Voter style ballot form with the candidate list and the associated right hand (teller) onion. His vote can now be cast in the usual way by marking an X against the candidate of their choice. The left hand strip is detached and discarded and the voter leaves the booth and casts their vote in the presence of an official exactly as described previously. Their receipt is recorded digitally as $(r, onion)$, where r is the index value indicating the position of the X .

The receipt can be digitally signed and franked at this point to counter any receipt faking attacks.

Once the election has closed, copies of the digitised receipts will be posted to the WBB exactly as before and the voters can visit this and assure themselves that their receipt has been correctly registered. In addition to this, a Verified Encrypted Paper Audit Trail mechanism could be deployed: at the time of casting, an extra paper copy of the receipt is made and retained in a sealed audit box. This can be used to independently check the correspondence with the receipts posted to the WBB.

7 Re-encryption/tabulation mixes

Our construction leads to ElGamal onions which appear to be well suited to being put through re-encryption mixes. However, the form of the ballot receipts means that this is not quite straightforward: in addition to the onion term we have the index value, in the clear as it were. An obvious approach would be to send the receipt terms through the mix re-encrypting the onions whilst leaving the index values unchanged. The problem with this is that an adversary is able to partition the mix according to the index values. There may be situations in which this is acceptable, for example large elections in which the number of voters vastly exceeds the number of voting options. In general it seems rather unsatisfactory.

A more satisfactory solution, at least for the case of a simple selection of one candidate from the list, is described in this section. We will discuss how to achieve full mixing in the more general case in section 10.

In this case we restrict ourselves to just cyclic shifts from the base ordering of the candidate list from a base ordering. For single candidate choice elections, this is sufficient to ensure that the receipts do not reveal the voter's selection. For more general styles of election, in which for example voters are required to indicate a ranking of the candidates, we of course need to allow full permutations of the candidate list. Indeed, even in the case of single selection elections, it is preferable to allow full permutations in order to eliminate any possibility of a systematic corruption of votes. For this moment we discuss the approach of simple cyclic shifts.

Let s_i be the shift of the candidate list for the i th ballot form. We can absorb the index value r into the onion:

$$(\alpha^y, \beta_T^y \cdot \gamma^{r-s_i})$$

This gives a pure ElGamal term and the value $r-s_i$ taken modulo n indicates the voter's the original candidate choice in the base ordering. These ElGamal terms can now be sent through a conventional re-encryption mix by a set of mix tellers, see for example [JJR02]. These mix tellers do not hold any secret keys but read in a batch of ElGamal terms from the WBB, re-encrypt each of them and then post the resulting terms in random order to the WBB. After an

appropriate number of such anonymising re-encryption mixes, (a threshold set of) the decryption tellers take over to extract the plaintext values.

Thus, in contrast to the decryption mixes uses previously, the anonymising and decrypting phases are separated out in re-encryption mixes.

This will yield decrypted terms of the form:

$$\gamma^{r-s_i} \pmod{p}.$$

Now we have to extract the values $r - s_i \pmod{n}$ to recover the original votes. The difficulty is that $r - s_i$ is the discrete log of γ^{r-s_i} in Z_p^* so in general, if the seed values had been drawn randomly from Z_p^* , computing this would be intractable. However, we have set things up so that the s values are drawn from a binomial distribution so we can search the space very efficiently. We could, for example, generate a look-up table for the logs out to some multiple of $\sigma\sqrt{l}$. Occasionally we will have an outlier that will require some search beyond the range of the look-up table.

7.1 Coercion resistance and plausible deniability

The point of using a binomial distribution for the seed value is to ensure plausible deniability or coercion resistance whilst at the same time avoiding the discrete log problem. An alternative approach would be to bound the possible seed values generated by the clerks to lie in some fixed range, between $-M$ and $+M$ say. This would have the problem that occasionally we would hit situations in which final decrypted $r - s$ values would take on extreme values, e.g., $r - s = -M$. In this case, an adversary could deduce that r must have equalled 0 and so be able to link this vote value back to a subset of the receipts, i.e., receipts with the index value 0.

Using a distribution avoids such “edge effects” whilst avoiding our having to compute arbitrary discrete logs in Z_p^* . Arguably, the adversary would be able to assign a non-flat probability distribution to the possible r values, but as long as no values of r can ever be eliminated, plausible deniability will be maintained.

We should also observe that even if it were possible to link a vote back to a particular index value, this would not typically violate ballot secrecy unless this it so happened that this identified a unique receipt, i.e., there happened to be only one receipt with this r value.

8 Auditing the Ballot Forms

The mechanisms described above allow for the distributed generation of ballot forms and just-in-time decryption of the candidate list and printing of the ballot forms. This has clear advantages in terms of removing the need to trust a single

entity to keep the ballot form information secret and avoiding chain of custody issues. On the other hand, it means that we can no longer use the random pre-auditing of pre-printed ballot forms as suggested in [CRS05]. Consequently, we must introduce alternative techniques to detect and deter any corruption or malfunction in the creation of the ballot forms.

A possible approach, in the supervised context at least, is to incorporate the two sided ballot form mechanism suggested in [Rya06] and re-introduce a cut-and-choose mechanism into the voter protocol. Here, a ballot form would be assigned two independent, entangled pairs of onions. One printed on one side of the form, the other on the flip side. In the booth, on each side, the left hand onion would be decrypted and the corresponding candidate list printed in the left hand column. The result is two independent ballot forms, one printed on each side, as illustrated in Figure 3.

Obelix		-----
Asterix		-----
Panoramix		-----
Idefix		-----
	<i>7rJ94K</i>	-----

Side 1

Panoramix		-----
Idefix		-----
Obelix		-----
Asterix		-----
	<i>Yu78gf</i>	-----

Side 2

Fig. 3. Prêt à Voter ballot form

Figure 3 shows the two sides of such a dual ballot form. These two sides should be thought of as rotated around a vertical axis. Note that each side has an independent randomization of the candidate order along with the corresponding cryptographic values. Thus each side carries an independent Prêt à Voter ballot form.

The voter uses only one side to encode their vote and makes an arbitrary choice between the sides. Suppose that the voter in this case chooses what we are referring to as side 2 and wants to cast a vote for Idefix. They place an X against Idefix on side 2 and then destroy the left hand strip that shows the candidate order for side 2. This results in a ballot receipt of the form shown in Figure 4.

These two sides should be thought of as being rotated around a vertical axis with respect to each other. Thus the shaded, third column of side 1 would oppose the candidate list of side 2.

The voter makes a random choice of which side to use to cast their vote and made their mark on the middle column against their candidate of choice and leave the flip, unselected side blank. The left hand column of the selected side is destroyed, and so the blank column of the flip side is destroyed. This results in a receipt on which the candidate list for the chosen side has been destroyed, whilst the ballot form on the slip, unselected side is intact, i.e., still has the onion value

Obelix	
Asterix	
Panoramix	
Idefix	
	<i>7rJ9AK</i>

auditable side
(remaining part of Side 1)

X	-----

<i>Yu78gf</i>	-----

vote encoding side
(remaining part of Side 2)

Fig. 4. Both sides of a Prêt à Voter ballot receipt

and candidate list. The information on both sides would now be recorded when the ballot is cast and posted to the WBB.

This flip side can now be audited and checked to ensure that the candidate list printed by the booth correctly corresponds to the onion value. Such checks could be performed immediately at the time of casting to detect any problems as soon as possible. Additionally, checks could be performed on the posted values.

In addition to such post-auditing of the dual ballot forms, we can do some pre-auditing of the committed onions pairs. This would help pick up any malfunctions or corruption in the preparation of the proto-forms at an early stage.

9 Auditing the anonymising mixes

In order to detect any malfunction or corruption by the mix tellers, we can again use the Partial Random Checking approach of [JJR02]. Here the checks on audited links will be slightly different: rather than revealing the seed information for the layer in question, the teller is required to reveal the re-randomisation value used to e-encrypt the select link. Auditing of the decryption tellers is quite straightforward as we don't need any further mixing at this stage (the anonymising mixes will be enough to ensure ballot secrecy). The correctness of the decryptions can thus be directly checked by simply encrypting the final values with the public keys and checking that these agree with the initial terms.

10 Handling full permutations and STV style elections

In order to deal with full permutations of the candidate list it is not immediately clear how to generalise the approach of section 7. As mentioned, one possibility is to leave the index values unchanged through the mixes. This might be acceptable in some situations but is clearly not satisfactory in general.

One solution is simply to have one onion for each candidate position. For a single candidate selection the ballot receipt would in effect simply be the onion

value against the chosen candidate. This feels rather inelegant and inefficient in terms of multiplying up the number of onions required.

For a ranked voting method, in which the voters are required to place a rank against each candidate, a ballot receipt would now comprise n pairs of rank value and onion. Each of these pairs could be put through the mix separately with the rank value unchanged (allowing the adversary to partition the mix according to the rank values seems not to matter). This approach works fine as long as the voting method does not require a voters rankings to be kept grouped for tabulation, as with STV for example.

11 Remote voting with Prêt à Voter

The encrypted ballot forms proposed here would appear to be adaptable to remote voting. We could for example, use a protocol like that described in [ZMSR04], to transform left-hand onions encrypted under the registrars' public key to terms encrypted under an individual voter's public key. The protocol of [ZMSR04] achieves this without having to reveal the underlying plaintext (seed) in the process. A pair of such ballot forms could be supplied to each voter in order to mimic the cut-and-choose mechanism described above. Details of such protocols are the subject of ongoing research.

Any remote voting scheme must face problems of coercion. A possible approach to counter such threats is the use *capabilities* as proposed in [JCJ02]. The possibility of using such a mechanism in conjunction with Prêt à Voter 2005 was explored in [CM05]. Voters are supplied with capabilities that are essentially encryptions of a nonce and a *valid* string. Votes are cast along with a capability and these go through the mix alongside the ballot terms. They emerge from the mix decrypted. A valid capability will decrypt to a valid plaintext. The validity or otherwise of the capability is not apparent until it is decrypted. As a consequence, a voter who is being observed whilst casting their vote has the possibility of deliberately and surreptitiously corrupting their capability. As long as the voter has some window of unobserved access to system he can cast his vote with his valid capability.

12 Conclusions

We have proposed some extensions to Prêt à Voter 2005 to counter vulnerabilities identified previously:

- Authority knowledge of ballot form crypto variables.
- Chain of custody threats.
- Chain voting attacks.

- Kleptographic channels.

The new version of the scheme counters these threats by enabling the distributed construction of encrypted ballot forms by a set of clerks. As a result, only a collusion of all the clerks could determine the cryptographic seed values. This eliminates the need to trust a single entity to keep this material secret and prevents Kleptographic attacks.

Our construction results in ballot forms in which the cryptographic seed values remain encrypted and can be decrypted on demand. Thus, the ballot forms with the candidate ordering can be created and printed in the booth, so eliminating chain of custody and chain voting threats.

The new construction uses ElGamal encryption and so is better suited to using re-encryption mixes for the anonymising/tabulation phase. Earlier work on robust ElGamal mixes may be found in [JJ01,Nef01,GJJS04]. The rather special representation of the ballot receipt in Prêt à Voter, index value plus cryptographic onion, means that it is not entirely straightforward to send such terms through a re-encryption mix. We have shown how, for single candidate selection and cyclic shifts of the candidate list at least, the ballot receipts can be transformed into pure ElGamal terms and so are adapted to re-encryption mixes. We have indicated how the approach may be generalised to deal with alternative electoral methods.

This version of the scheme is, we believe, technically superior to the 2005 version in that it requires less trust assumptions and is more robust against a number of threats. On the other hand, from a socio-technical point of view, it may have certain disadvantages. The voter experience is a little more complex, in particular the need for the cut-and-choose element on the voter protocol, which could have usability implications as well as opening up possibilities of “social engineering” style attacks, [KSW05]. Thus, it is possible that, for some situations like general elections perhaps, in evaluating the trade-off between the trust assumptions of Prêt à Voter 2005 and the usability issues of this scheme, the former might be deemed more acceptable.

13 Acknowledgements

The authors would firstly like to thank Paul Syverson and Ron Rivest for suggesting adapting Prêt à Voter to work with re-encryption mixes. We would also like to thank James Heather for many helpful discussions, as well as Ran Cannetti, Michael Clarkson, Rob Delicata, Joshua Guttman, Markus Jakobsson, and Thea Peacock. Finally, we are grateful to Paul Syverson and the anonymous referees for their comments on this paper.

References

- [CM05] M. Clarkson and A. Myers. Coercion-resistant remote voting using decryption mixes. In *Workshop on Frontiers of Electronic Elections*, 2005.
- [CRS05] D. Chaum, P.Y.A. Ryan, and S. Schneider. A practical, voter-verifiable election scheme. In *European Symposium on Research in Computer Security*, number 3679 in Lecture Notes in Computer Science. Springer-Verlag, 2005.
- [GJJS04] P. Golle, M. Jakobsson, A. Juels, and P. Syverson. Universal re-encryption for mixnets. In *Proceedings of the 2004 RSA Conference, Cryptographers' track*, San Francisco, USA, February 2004.
- [JCJ02] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant electronic elections. To appear, 2002.
- [JJ01] M. Jakobsson and A. Juels. An optimally robust hybrid mix network (extended abstract). In *Proceedings of Principles of Distributed Computing — PODC'01*. ACM Press, 2001.
- [JJR02] A. Jakobsson, A. Juels, and R. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In *USENIX Security '02*, 2002.
- [KSW05] C. Karlof, N. Sastry, and D. Wagner. Cryptographic voting protocols: A systems perspective. In *USENIX Security Symposium*, number 3444 in Lecture Notes in Computer Science, pages 186–200. Springer-Verlag, 2005.
- [M. 06] M. Gogolewski et al. Kleptographic attacks on e-election schemes. In *International Conference on Emerging trends in Information and Communication Security*, 2006. <http://www.nesc.ac.uk/talks/639/Day2/workshop-slides2.pdf>.
- [Nef01] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM Conference on Computer and Communications Security — CCS 2001*, November 2001.
- [RP05] P.Y.A. Ryan and T. Peacock. Prêt à voter: a systems perspective. Technical Report CS-TR-929, University of Newcastle upon Tyne, 2005.
- [Rya06] P.Y.A. Ryan. Putting the human back in voting protocols. In *Fourteenth International Workshop on Security Protocols*, Lecture Notes in Computer Science. Springer-Verlag, 2006. To appear.
- [YY96] A. Young and M. Yung. The dark side of black-box cryptography, or: Should we trust capstone? In *Crypto'96*, Lecture Notes in Computer Science, pages 89–103. Springer-Verlag, 1996.
- [ZMSR04] L. Zhou, M. Marsh, F. Schneider, and A. Redz. Distributed blinding for elgamal re-encryption. Technical Report TR 2004-1920, Cornell University, January 2004.

Part Socio – APPENDIX
(Resilient Socio-Technical Systems)

Automation Bias in Computer Aided Decision Making in Cancer Detection: Implications for System Design

E. Alberdi*, P. Ayton[†], A. A. Povyakalo*, L. Strigini*

*Centre for Software Reliability, City University, London, UK

{e.alberdi, andrey, strigini}@csr.city.ac.uk

[†] Psychology Department, City University, London, UK

P.Ayton@city.ac.uk

Keywords: decision support, computer aided decision making, alerting systems, human-machine diversity

Abstract

In this paper we summarise the results of an interdisciplinary study of computer-aided decision making in cancer detection. Our study included probabilistic modeling, refined statistical analyses of evaluation data from an independent controlled study and an empirical study of the effects of incorrect computer output on the decisions of expert clinicians. Our results strongly suggest that, at least for some categories of cases, incorrect computer output had a significant detrimental effect on human decisions. We discuss these results in the context of the human factors literature on “automation bias” (roughly, negative effects of over-reliance on automation), which we review extensively in this paper. Automation bias effects have not been previously reported for this application with expert clinicians, and are not generally considered in the medical/radiological literature. One lesson from our study was that these potentially important effects will easily go unnoticed with common assessment methods. Implications for designers of computer aided decision making include that: a) it may be necessary to calibrate tool design for a range of different levels of user skills; b) an “expert modeling” approach to computerised aid design – building a “better replica” of a human expert – may be counterproductive by making the aid weak in those very areas where humans need help; c) HCI design risks focusing on usability of the physical human-computer interface, but the critical issues in design concern the cognitive effects (e.g. changes of decision thresholds) a computer tool may have on users; d) users’ subjective assessments of a computer aid may be misleading: people may judge a tool helpful when their decision-making performance is actually being hampered.

1 Introduction

This paper deals with *computer-aided decision making*, where automated aids support human decisions with advice, filtered or enhanced information, alerts and prompts. From the viewpoint of design for dependability, computer aided decision making offers *diverse redundancy* (Littlewood, Popov et al. 2002; Strigini 2005), with diverse components (people and machines) expected to detect and support the correction of errors of other components.

Designing such *human-computer* systems involves attending to a number of different aspects of the system’s components. Their dependability is affected by the design of the decision support tools, the procedures for their deployment and use and the training of operators and the definition of their roles.

The intended role of automation is often purely auxiliary: to reduce the risk of humans missing some information (e.g. when fatigued or overloaded), or to make the process of human apprehension of all important information more efficient. The human user retains the authority and responsibility for

decisions. The design intention is thus to exploit *protective redundancy* with *diversity*: the machine protects against some human errors, and vice versa.

We summarise here analyses and empirical work we conducted of a particular application of computer aided decision making: Computer Aided Detection (CAD) in mammography, i.e., clinicians using computer support in reading X-ray images for breast cancer screening.

The study, within the context of DIRC (the Interdisciplinary Research Collaboration on the Dependability of computer-based systems) used an interdisciplinary approach, combining insights and methods from reliability engineering, psychology, human factors and ethnography (Alberdi, Povyakalo et al. 2005). It was a follow-up to a previous controlled study of CAD conducted by other researchers for the UK Health and Technology Assessment (HTA) programme, on a specific CAD tool ("Study 1" in (Taylor, Champness et al. 2005), hereon "the HTA study").

Our probabilistic modelling (Strigini, Povyakalo et al. 2003), inspired by previous experience on software systems with diverse redundancy (Littlewood, Popov et al. 2002), highlighted how variation and co-variation in the "difficulty" of input cases for the machine and human components of the system substantially affect the dependability of the overall system: focusing on the average probabilities of the failures of the components, or assuming statistical independence among their failures, can be misleading. We were granted access to raw data from the HTA study, on which we conducted various exploratory statistical analyses, focusing on the interactions between correctness of computer output, difficulty of individual decision problems and human skills. Additional empirical work by us and our DIRC colleagues included follow-up experiments (focusing on the effects of incorrect computer output on decisions) and ethnographic observation, both of which helped to elucidate aspects of the users' behaviour.

Our statistical results (Alberdi, Povyakalo et al. 2004; Povyakalo, Alberdi et al. 2004), which we summarise here, suggested that the effects of computer assistance varied markedly across users and over the set of cases examined. In particular, the effects were shown to be potentially detrimental for some clinicians when dealing with some categories of cases. This finding was striking in this application context because it refuted an assumption often found (implicitly or explicitly) in discussions of the efficacy of CAD, i.e., that the *computer aid can do no harm* (cannot cause bad decisions); moreover, this assumption has implications for the methods to be used in estimating the tool's efficacy. However, the observation of varying effects of computer aids on human decisions is more generally important as a basis for conjecturing *how* these effects are generated, and thus informing better design of decision systems. We discuss here possible explanations for the observed detrimental effects. It is tempting to categorise them all as effects of "complacency": human operators becoming less attentive or less wary of error since they can fall-back on the computer for decisions. But this would be simplistic and unfair. When experts, giving every indication of being attentive and thorough, appear to be led into error, it is prudent to contemplate other possible causes of error besides complacency, and to examine their implications for design and assessment practices in all applications of computer-aided decision-making.

Our aim is to discuss these general implications, *not* the effectiveness of CAD in general or of the specific tool in particular: both issues are widely debated in the medical literature and CAD tools evolve rapidly so that data obtained about one version may rapidly become obsolete (BJR 2005).

In the rest of this paper, we review some pertinent literature (Section 2), and introduce the application domain of our study (Section 3). Section 4 describes our exploratory statistical analyses of the raw data from the HTA study. In Section 5, we outline the results of our follow up empirical studies, as well as evidence from questionnaires and ethnographic observations. Next we discuss potential explanations of the behavioural patterns observed (Section 6) and their implications for the design of decision systems (Section 7). Section 8 contains our conclusions.

2 Automation bias literature

"Automation bias" refers to those situations where a human operator makes more errors when being assisted by a computerised device than when performing the same task without computer assistance (Mosier, Skitka et al. 1998; Skitka, Mosier et al. 1999). Similar or related concepts are automation-

induced “complacency” (Azar 1998; Singh, Molloy et al. 1993; Wiener 1981), and “overreliance” on automation (Parasuraman and Riley 1997).

This review focuses on computer assisted monitoring or decision making, where an automated aid supports the human decisions with advice, filtered or enhanced information, alerts and prompts (as is the case with CAD for mammography, see Section 3). These human-machine decision systems correspond mostly to what Parasuraman et al. (Parasuraman, Sheridan et al. 2000) classify as Stage 1 and Stage 2 automation: Stage 1 automation refers to automated tools that support the human by filtering or focusing attention on information deemed of interest and Stage 2 automation refers to tools that support the human by forming inferences of the state of the world or by integrating information. In these types of human-computer systems, the final decision and action are the responsibility of the human operator. Examples of computer tools in these categories are: warning devices, medical diagnostic or detection tools, intelligent target aiding, collision alerts and statistical tests.

(Stage 3 and Stage 4 in Parasuraman et al.’s (Parasuraman, Sheridan et al. 2000) classification correspond to higher levels of automation where the computer tool either recommends/selects a course of action or implements the action).

Although many studies support the view that the collaboration between automation and a human decision maker can be beneficial and effective (Corcoran, Dennett et al. 1972; Dalal and Kasper 1994; Parasuraman 1987; Thackray and Touchstone 1989), it has long been recognised that human-computer “decision systems” do not always function ideally (Bainbridge 1983; Sorkin and Woods 1985). For example, Sorkin & Woods (1985), using signal detection analysis, concluded that optimising an automated aid’s performance would not always optimise the performance of a human-computer team in a monitoring task. The implication is that human-machine system can only be optimised or improved as a whole: improving the automation alone or human training alone may be ineffective or wasteful.

The view that automation simply replaces or supports the human operator is too simplistic and sometimes incorrect as automation fundamentally changes the nature of the cognitive task that the human operator does, often in ways that were not intended or anticipated by the developers of automated tools (Parasuraman and Riley 1997).

2.1 Basic concepts and terminology

Parasuraman and Riley (Parasuraman and Riley 1997) discussed ways in which human-computer interaction can go wrong, presenting anecdotal evidence and results from various empirical studies. They talked about three aspects of ineffective human use of automation:

- *disuse*, i.e., underutilization of automation, where humans ignore automated warning signals;
- *misuse*, i.e., overreliance on automation, where humans are more likely to rely on computer advice (even if wrong) than on their own judgement;
- *abuse*, when technology is developed without due regard for human needs or the consequences for human (and hence system) performance and the operator’s authority in the system.

Skitka, Mosier and Burdick (Skitka, Mosier et al. 1999) focused on the *misuse* of automation, in particular on the “automation bias” effects occurring when people used wrong computer advice for monitoring tasks in aviation. They distinguished two types of error:

- *errors of commission*: decision-makers follow automated advice even in the face of more valid or reliable indicators suggesting that the automated aid is wrong;
- *errors of omission*: decision makers do not take appropriate action, despite non-automated indications of problems, because the automated tool did not prompt them.

Focusing on *warnings* generated by automated tools, Meyer (Meyer 2004), distinguishes between two alternative ways in which humans can “follow the advice” from a warning system : compliance and reliance:

- *compliance* indicates that the operator acts according to a warning signal and takes an action.
- *reliance* is used to describe those situations where the warning system indicates that “things are OK” and the operator accordingly takes no action.

As a result, in Meyer's terms, undue *compliance* (complying with an incorrect automated warning) would lead to *errors of commission* and undue *reliance* (failing to take action when no automated warning is issued) would lead to *errors of omission*.

2.2 Studies of automation bias

The phrase “automation bias” was introduced by Mosier et al. (Mosier, Skitka et al. 1998) when studying the behaviour of pilots in a simulated flight. In this study, they encountered the errors of omission and commission described above. These findings were subsequently replicated in studies using non-pilot samples (student participants) in laboratory settings simulating aviation monitoring tasks (Skitka, Mosier et al. 1999). Essentially they found that, when the automated tool was reliable, the participants in the automated condition made more correct responses. However, with automation that was imperfect (i.e. unreliable for some of the tasks), people in the study were more likely to make errors than those who performed the same tasks without automated advice. For the monitoring tasks that Skitka and colleagues used in their studies, the decision-makers had access to other (non automated) sources of information. In the automated condition they were informed that the automated tool was not completely reliable but all other instruments were 100% reliable. Still, many chose to follow the advice of the automated tool even when it was wrong and was contradicted by the other sources of information. The authors concluded that these participants had been biased by automation and interpreted their errors (especially their errors of omission) as a result of complacency or reduction in vigilance.

As indicated above, people's ineffective use of computerised tools is often described in terms of “complacency”, resulting from overreliance on automation (Azar 1998; Singh, Molloy et al. 1993; Wiener 1981). Singh et al. (Singh, Molloy et al. 1997) quote the ASRS Coding Manual's definition of “complacency” as “self-satisfaction which may result in non-vigilance based on an unjustified assumption of satisfactory system state” (Billings, Lauber et al. 1976).

Similarly, in a review of studies of pilot use of aviation-related decision support systems, Cummings (Cummings 2004) proposes that some automation bias effects can be explained in terms of “confirmation bias”, that is, people's tendency to disregard or not search for information that contradicts a belief they already formed or solution they chose, in this case the belief or solution generated by a computer. This tendency can be exacerbated for time critical tasks.

A problem with the term “complacency” (and similar accounts of automation bias) is that they suggest value judgments on the human experts. As Moray (Moray 2003) has recently pointed out, the claim that automation fosters complacency implies that operators are at fault, when the problem often lies in the characteristics of the automated tools, not in the human operators' performance; hence he advocates for a radical re-design of monitoring tools.

Similarly, in a recent review of human use of diagnostic automation, Wickens and Dixon (Wickens and Dixon 2005) question the notions of complacency or reduced vigilance as explanations of automation bias phenomena. Instead, they argue that operators, whilst being aware of the unreliability of the diagnostic tools, choose to depend on the imperfect computer output to preserve available processing resources for other tasks, particularly in situations with high workload. The authors conjecture that, even in “non-high workload” situations, the human operator may have an inherent need to protect some reserve capacity for unexpected tasks that may arise.

Individual differences seem to play an important role in human reactions to automation (Skitka, Mosier et al. 1999; Skitka, Mosier et al. 2000; Dzindolet, Peterson et al. 2003). One would expect that more experienced people would tend to be less susceptible to bias from automation when performing tasks in their area of expertise. However, as highlighted above, automation bias effects have been reported for both laymen (e.g. student participants) and experts (e.g. air pilots and clinicians). In fact, Galletta et al. (Galletta, Durcikova et al. 2005) have shown (in a verbal skills domain using spelling and grammar

checking software) that more skilled and experienced individuals were more likely to be damaged by certain types of computer errors (e.g. false negatives) than were the less skilled individuals.

Factors that have been proposed as possible influences in people's vulnerability to automation bias include:

- people's accountability for their own decisions (Skitka, Mosier et al. 2000);
- the levels of automation at which the computer support is provided (Meyer, Feinschreiber et al. 2003; Cummings 2004);
- the location of computer advice/warnings with respect to raw data or other non-automated sources of information (Meyer 2001); (Singh, Molloy et al. 1997).

2.3 Trust in automation

Some researchers have discussed human reliance on automation in terms of *trust* (Bisantz and Seong 2001); (Dassonville, Jolly et al. 1996; Dzindolet, Peterson et al. 2003; Lee and Moray 1994; Lee and See 2003; Muir 1987; Muir 1994; Muir and Moray 1996; Parasuraman and Riley 1997; Singh, Molloy et al. 1993; Tan and Lewandowsky 1996). The idea is that human operators are the more likely to rely on an automated aid the more they trust the aid. If a human trusts an aid that is adequately reliable or fails to trust an aid that is indeed too unreliable, appropriate use of automation should occur as a result. However if a human trusts (and therefore relies on) an unreliable tool, then automation bias may occur (or *misuse* of automation as defined above). Similarly if a person does not trust a highly reliable tool, the person may end up *disusing* (as defined above) or under-using the tool, hence the full potential benefits of automation will not be fulfilled.

Indeed, subjective measures of the trust of human operators in a computer tool have been found to be highly predictive of people's frequency of use of the tool (de Vries, Midden et al. 2003; Dzindolet, Peterson et al. 2003). Use of automation (or *reliance* in automation in its generic sense) is usually assessed with observations of the proportion of times during which a device is used or by assessing the probability of detecting automation failures (Meyer 2001).

A concept related to "trust" in automation is that of the "credibility" or "believability" of automation (Tseng and Fogg 1999). There are indications that (some) people tend to perceive computers as infallible, and may put excessive trust in them (Fogg and Hsiang 1999; Martin 1993). Indeed, (Dzindolet, Peterson et al. 2003) have shown empirically that people have an inclination to trust and rely on an automated aid regardless of its reliability. However, there is also evidence that as soon as humans become aware of the errors made by a computer tool, their trust in the tool, and their subsequent reliance on it, decrease sharply (de Vries, Midden et al. 2003; Dzindolet, Peterson et al. 2003; Dzindolet, Pierce et al. 2002). This reduction in trust, in turn, can be attenuated by increasing people's understanding of the computer errors. Dzindolet et al. (Dzindolet, Peterson et al. 2003) found that people who were told the reasons for the aid's errors were more likely to trust it and follow its advice than those who were not aware of these reasons.

In addition to people's perception of the reliability of an automated tool, other factors will influence trust in automation. For example, (Parasuraman and Miller 2004) define an automated tool as having good "etiquette" if it is not invasive or is not perceived as "interrupting" the user or as being "impatient". They found that automated tools with "good etiquette" (in this sense) were more likely to be trusted by human operators and led to better human performance even in high-criticality automation. Similar results have been found in studies of the invasiveness of collision alarms in automobiles (Bliss and Acton 2003).

3 Domain: CAD for mammography

In screening for breast cancer, expert clinicians ("readers") examine *mammograms* (sets of X-ray images of a woman's breasts), and decide whether the patient should be "recalled" for further tests because they suspect cancer.

A CAD tool is designed to assist the interpretation of mammograms. It aims to recognise and mark (“prompt”) regions of interest (ROI) on a digitized mammogram to prevent clinicians from overlooking them. CAD is not meant to be a diagnostic tool, in the sense that it only marks ROIs, which should be subsequently classified by the reader to reach the “recall/no recall” decision.

The CAD tool in our study was the R2 Imagechecker M1000 (FDA 1998). The prescribed procedure for using it is: the reader looks at a mammogram and interprets it as usual, then activates the tool and looks at a digitised image of the mammogram with the tool's prompts for ROIs, checks whether he/she has overlooked any features with diagnostic value and then revises his/her original assessment, if appropriate. Fig. 1 schematically shows the resulting “decision system”.

The manufacturers claim that when this tool is used as recommended the potential for missing lesions is not increased over routine screening (FDA 1998).

This decision system may produce two kinds of errors: *false negative* (FN: the reader issues a “no recall” decision on a cancer) or *false positive* (FP: recalling a non-cancer – a *normal* – case). In turn, we define a FN error by the CAD tool as failing to prompt areas of the mammogram where cancer is located, even when the tool places prompts on other areas (we also call a FN error “incorrect prompting” of a cancer), and a FP error as placing *any* prompt on the mammogram of a normal case.

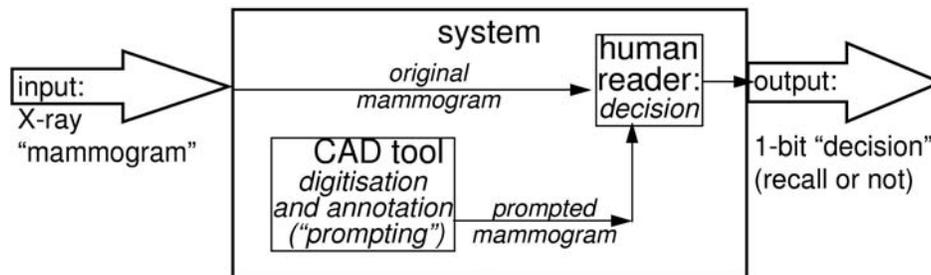


Figure 1. The decision system formed by a human “reader” with the Computer Aided Detection tool.

4 Interactions between correctness of computer output, difficulty of cases and human skills

In the HTA study (Taylor, Champness et al. 2005), 50 readers looked at mammograms for which the correct diagnosis was known (60 cancer and 120 normal cases) in two conditions: with and without computer support. Each reader examined every case, once in a session with computer aid and once in a session without it (in independent randomised orders) producing a “recall/no recall” decision. This study found no statistically significant effect of CAD prompts on the performance of these readers in detecting cancers.

We re-analysed the raw data from this study looking especially for effects that may not be visible in the average results. These supplementary analyses (Alberdi, Povyakalo et al. 2005) indicated that:

1. correct computer prompting was likely to help readers in reaching a correct decision while incorrect prompting hindered them;
2. even without computer support, incorrect decisions were more likely for cases that the CAD tool also processed incorrectly: reader and computer errors are strongly correlated. Consequently, for those cases that are more difficult for humans to interpret, the computer is less likely to give useful output.

We define the *difficulty* of a case as the fraction of readers who produced an incorrect decision about that case. We call the difficulty of a case read with computer aid (prompting) dp and the difficulty without computer aid d . Statistical analyses indicated that the value: $d - dp$ was significantly different between correctly and incorrectly prompted cases (both for cancers and normal cases).

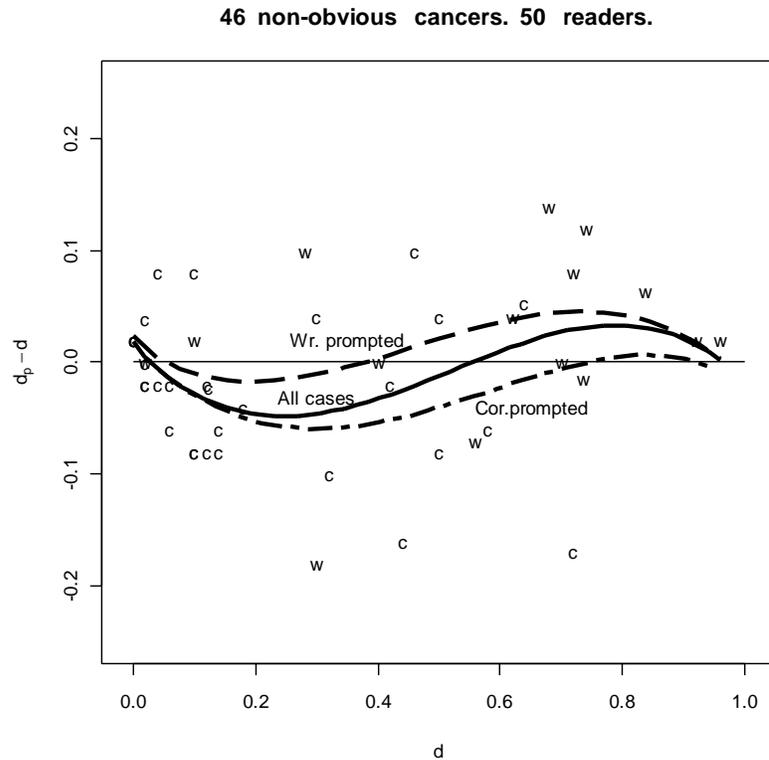


Figure 2. Effect of correctness of CAD prompting on case difficulty (i.e., rate of FN errors): dp : difficulty of a case with computer aid; d : difficulty without computer aid; w : cancer incorrectly prompted by CAD; c : cancer correctly prompted by CAD; dashed curve: logistic regression curve for wrongly processed cancers; dashed-dotted curve: logistic regression curve for correctly processed cancers; solid curve: logistic regression curve for all cancers (Povyakalo, Alberdi et al. 2004).

We also used logistic regression (a form of interpolation on the raw data) to look for possible general patterns in the effect of CAD (Povyakalo, Alberdi et al. 2004). This showed that CAD tended to make cancers that were relatively easy (i.e. with $d < 0.6$) less difficult (i.e., $dp < d$), and cases which were relatively difficult (i.e. with $d > 0.6$) even more difficult (i.e., $dp > d$).

Fig. 2 illustrates this effect. The regression analysis covered "non-obvious" cancer cases, that is, cancers missed by at least one reader in either condition. The horizontal axis represents their difficulty d without computer aid; the vertical axis shows the differences ($dp - d$). So, a point below 0 on the y-axis indicates a cancer for which CAD appears to reduce the rate of FN errors. Points marked 'w' and 'c' indicate the observed values of d and ($dp - d$) for non-obvious cancers, divided into those with correct CAD output ('c' symbols) and with wrong CAD output ('w' symbols). The curves show the regression estimates for the mean value of ($dp - d$) as a function of the difficulty d of cases. The dashed curve is obtained from the data for the incorrectly prompted cancers, the dotted-dashed curve to the correctly prompted cancers and the solid curve to all cancers together.

Analyses of all readers' decisions without computer aid, suggest that the more *sensitive* readers (those more likely to decide correctly on cancers) recognised more of the difficult cancers than other readers. Fig. 2 shows that, for difficult cases, incorrect decisions were more common with computer aid than without. Since less sensitive readers were very unlikely to recognise difficult cancers anyway, this

increase in FN errors for difficult cases must be due to an adverse effect of the computer aid on the decisions of the more sensitive readers, plausibly caused by incorrect computer prompts (cf Fig 2, again). Similarly, for the "easy" cases, without computer aid the less sensitive readers made a larger number of incorrect decisions: they were thus more likely to benefit from the correct computer prompts on "easy" cases than on "difficult" ones, and on easy cases they were more likely to benefit than the more sensitive readers.

Thus, the use of computer support is likely to be more beneficial for less sensitive readers (i.e., it may improve their sensitivity) and more questionable for the more sensitive ones (as it can decrease their sensitivity for some cases). This conjecture is supported by additional statistical analyses of these data (Povyakalo, Alberdi et al. 2005).

Apart from evaluating the probable effect of CAD in future use (Strigini, Povyakalo et al. 2003; Alberdi, Povyakalo et al. 2005), these statistical patterns (and observed exceptions to them) can help identify possible mechanisms causing improved or worse decisions. Examples of exceptions can be seen in Figure 2: despite being prompted correctly (points highlighted as 'c'), there were a few "easy" cancer cases (with $d \leq 0.1$) missed by more readers with computer support than without it. For instance, one cancer case was missed by 2 of 50 readers without computer aid ($d = 0.04$) and by 6 of 50 readers with computer aid ($dp = 0.12$). Such changes in decisions might be due simply to natural "noise" – random variation – in human decision making. But if the marked change observed here were a confirmed pattern, we would need to explore possible causes. For this particular cancer case, without CAD, 13 of 50 readers (26%) decided that this case required discussion before the final decision. This suggests that there was a relatively high level of uncertainty associated with this case, although the low number of FN errors in deciding about it define it as comparatively "easy". As we report below, evidence from ethnographic work and from our follow-up studies supports the conjecture that when dealing with uncertain ("indeterminate") cases, readers' CAD prompts can have unanticipated (and potentially detrimental) effects on readers' decisions.

5 Follow-up empirical studies: effects of incorrect computer output

We conducted two follow up studies, investigating in more detail the effects of incorrect computer output on human decision making (Alberdi, Povyakalo et al. 2004). These were complemented by the collection of subjective data via questionnaires.

5.1 Follow-up studies

In our follow-up Study 1 we used a test set containing a large proportion of cancers that CAD had missed. We kept all other characteristics of the test set as similar as possible to the sets used in the HTA study as we wanted the readers to perceive this study as a natural extension of the HTA study (all 20 readers in Study 1 had also participated in the HTA study) and to behave in a comparable way. We used essentially the same procedures used in the HTA study, except that the readers in Study 1 saw all the cases only once: always with the benefit of CAD.

At that stage, we were not interested in comparing readers' performance with and without CAD; our goal was to estimate the probability of reader error when the CAD tool's output was incorrect. However, we obtained intriguing, unexpected results: the proportion of correct decisions was very low, and particularly so for cancer cases not marked by the tool. This suggested that CAD errors may have had a significant negative impact on readers' decisions. To explore this issue further, we ran Study 2, where a new but equivalent set of readers saw the same test set without computer aid. We used the same procedure as in Study 1, except that readers did not see the computer prompts.

Reader sensitivity in Study 2 (without CAD) was significantly higher than in Study 1 (with CAD); the measured difference increased if we looked at incorrectly prompted cases only, and even more so for cancer mammograms where the tool had placed no prompts at all.

These findings strongly suggest that, at least for some categories of cases in our studies, incorrect prompting had a significant detrimental effect on human decisions. This conclusion reinforces that of a

previous experiment (Zheng, Ganott et al. 2001) which used artificially inserted errors in computer prompts, as opposed to the actual errors used in our study.

5.2 Questionnaire results

At various stages during Study 1, the readers were asked to answer a series of questionnaires on various issues. Of particular interest was a final questionnaire which probed the perceived differences between the test sets used in Study 1 and in the HTA study, in terms of the behaviour of the CAD tool and the characteristics of the test cases. Since only 10 out of the 20 readers in Study 1 completed this questionnaire, our analysis is limited. However, we found some interesting patterns.

Some readers seemed unaware of the large proportion of cancers for which the CAD tool generated incorrect prompting in our test set. For example, most of the respondents answered that our test set had the same proportion of cancers, or fewer, than the sets in the HTA study, when it actually contained a few more cancers. Similarly a few respondents thought our test set had the same proportion of cancers missed by the CAD tool as in the HTA study; but, in fact, it contained many more. Also, half of the respondents thought that the CAD prompts were more useful in Study 1 than in the HTA study because, they said, our test set had fewer “distracting” prompts; yet, in reality, there were fewer correct prompts for cancers.

Interestingly, we found no association between a reader's responses to the questionnaires and his/her performance in Study 1. Readers whose answers indicated a realistic perception of the behaviour of the CAD tool and of the proportion of cancers in the test set were as likely to be affected by incorrect prompts as those who believed CAD was being more helpful in Study 1 than in the HTA study.

6 Does CAD bias readers' decision making?

The results of our analyses and empirical studies suggest that the CAD tool's incorrect output may have biased the decision making of at least some of the readers, for some categories of cases. Ethnographic work by DIRC colleagues has shown that readers: a) dismissed obvious incorrect prompts as spurious (indeed, most prompts on features that the reader had not considered – either missed or thought irrelevant – *are* false prompts); and b) recognised that the tool often fails to prompt obvious cancers (Hartwood, Procter et al. 2003; Alberdi, Povyakalo et al. 2005). However, our results indicate that, for difficult cancers, incorrect prompting often led to incorrect human decisions.

Using Skitka et al.'s (Skitka, Mosier et al. 1999; Skitka 2000) terminology, introduced earlier, one could characterise these incorrect decisions as “errors of omission”: readers interpreted absence of prompts on (an area of) a mammogram as an indication of “no cancer” and therefore failed to take appropriate action (i.e., they failed to recall the patient). There are, however, important differences between Skitka et al.'s tasks and mammogram reading as investigated in the HTA study and our follow-up studies. In the former, the participants seemed to use the computer output to replace calculations they could perform otherwise by using alternative, highly reliable, non-computer mediated indicators. In contrast, uncertainty in mammogram reading is greater. The readers did not have access to any sources of information other than the X-ray films (mammograms) and the output of the CAD tool on digitised versions of the films. It is important to remember that CAD prompts are designed merely as attention cues - not to replace other sources of information.

With CAD, over-reliance on automation could manifest itself in readers using prompts instead of thoroughly examining the mammograms, implying readers becoming complacent, or less vigilant, when using automation – as hypothesised by Skitka's team and others to explain errors of omission (Parasuraman and Riley 1997; Skitka, Mosier et al. 1999; Skitka 2000; Meyer 2001; Meyer and Bitan 2002; Meyer, Feinsreiber et al. 2003; Meyer 2004; Parasuraman and Miller 2004).

One could argue that, based on past experience with the tool, readers tended to assume that the absence of prompts was a strong indication that a case was normal. CAD generates many false positives, which readers claim to find distracting. As a result, the absence of prompts may be seen as more informative than their presence so readers may have paid less attention than necessary to those mammograms that had no computer prompts.

An alternative (perhaps complementary) way of accounting for the association between incorrect prompting and reader errors contemplates how readers deal with “indeterminate” cases: cases where they have detected anomalies with unclear diagnosis, so that they are uncertain as to whether the cases should be recalled. For example, it is possible that the absence of prompts made readers revise their decisions for ambiguous abnormalities they *had* already detected. In other words, they may have used the absence of prompts as reassurance for a “no recall” decision when dealing with features they found difficult to interpret. Conceivably readers were using any available evidence to resolve uncertainty. The implication is that the CAD tool was being used not only as a detection aid but also as a classification or diagnostic aid – specifically *not* what the tool is designed for. Readers’ use of CAD in such a way has been reported before for the CAD tool investigated here (Hartwood, Procter et al. 2003) and for other similar CAD tools (Hartwood, Procter et al. 1998). The following transcript is an example of a reader’s use of prompts to inform her/his decisions: “This is a case where without the prompt I’d probably let it go ... but seeing the prompt I’ll probably recall ... it doesn’t look like a mass but she’s got quite difficult dense breasts ... I’d probably recall.” (Hartwood, Procter et al. 2003) In other instances, readers were observed using the absence of a prompt as evidence for ‘no recall’ (Hartwood, Procter et al. 2003)

Arguably, this use of CAD violates an explicit warning in the device labelling (FDA 1998): “...a user should not be dissuaded from working up a finding if the device fails to mark that site”. However, such warnings could only be useful if readers were aware of exploiting prompts in this fashion.

An alternative conjecture about “indeterminate cases” posits that CAD may alter readers’ decision thresholds. The conjecture is as follows: when seeing certain “indeterminate cases” without computer support, readers are likely to recall it for further investigation. However, with computer support, they know that supplementary information is available in the form of CAD prompts; therefore, their preliminary decisions (before checking the prompts) are more likely to be “no recall” than they would be without CAD (a similar point was previously raised about other studies (Astley and Gilbert 2004; Taylor, Given-Wilson et al. 2004)). In practice, this makes the computer outputs a determinant for the “recall/no recall” decision, despite readers *never* changing a preliminary decision from “recall” to “no recall” based on absence of prompts, or being in violation of the prescribed procedures. For those indeterminate cases that are “difficult” cancers, CAD is likely to produce incorrect output, substantially reducing the chances of the case being recalled. Readers may not change any individual decisions because of absence of prompts, but they may change their decision thresholds for some cases due to the very presence of computer support. It may be very difficult (or even impossible) to prevent this behaviour by simple prescription.

7 Implications for design

The decision system whose dependability we are considering is formed by the human reader plus an “alerting” tool (the CAD tool) - a common type of system, in many medical and non-medical applications. The people involved in designing such systems include the designers of the computer tool proper, those responsible for the client-specific configuration of the tool and those who devise procedures for its use and the training of the staff using and servicing it. What are the implications for these people of the possible subtle effects of computer use?

In this section we focus on the risk of FN failures: not raising an alarm when an alarm is required. For CAD, this means not recalling a patient with cancer. A false negative, in these decision systems, is usually a more severe failure than a false positive. Space precludes discussion of the major concern of achieving a low enough FN rate without an excessive FP rate – spurious alarms. However, there are many application scenarios when we can expect a degree of homeostasis in the FP rate: the human operator spontaneously limits the total rate of alarms to a level that is acceptable in terms of extra workload on the organisation. In these cases, success in reducing FN rates does not automatically translate into excessive FP rates.

It can be seen that the probability of incorrect decisions from a system like the one described is determined not only by the probability of failure of the CAD tool and of the reader separately, but also by how likely they are to fail together (for a mathematical treatment see (Strigini, Povyakalo et al. 2003)). In the experiments we described, there was indeed a high correlation between their failures. This

may be due to two factors: the same mammograms that are hard to interpret correctly by readers are also hard for the tool to prompt correctly; or, incorrect prompting by the CAD tool makes readers more likely to err in their turn. The data from the HTA study suggest that both factors were present.

Thus, multiple lines of attack are open to designers seeking to improve this alarm system. An obvious one is to improve either component – the tool or its user. But improvements in algorithms and/or in user training may be difficult and subject to a law of diminishing returns. They could also be comparatively ineffective, if for instance we improve a CAD tool's ability to prompt kinds of cancers that readers very seldom miss anyway. Other strategies, targeting the interaction and correlation between machine and human error, may be more attractive. We outline two general categories of such strategies, some of them already adopted by designers, others possibly more novel.

7.1 Make alerting tools more different from their users

The first strategy aims to reduce the covariance between the difficulty of a case for the alerting tool and for the unaided user: making the tool less likely to err in those situations in which the unaided user is more likely to err. This strategy is feasible, since the designers of an alerting tool have a degree of freedom in choosing the trade-off between its FP and FN rates. Especially if the tool uses a combination of algorithms for identifying situations that require an alert, tuning these multiple algorithms may allow some "targeting" of the peaks and troughs in FN and FP rates for different classes of "cases".

An additional desirable effect of this strategy may be the scope for designers to reduce the overall FP rate of the tool: users find FPs annoying and they may increase cognitive load. For some applications, it may even be possible to tune the alerting tool for each individual user, tuning which could be under control of the users themselves or achieved via automatic calibration routines.

The importance of the probability of common failure between human and alerting tool may have more general implications for the design philosophy for these tools. A design philosophy aiming at reproducing the outward behaviour of human experts may bring intrinsic limitations to the effectiveness of these tools, in that they will tend to help the user most reliably on those cases where the user needs less help. They will still be immune to fatigue and random lapses of attention, and thus serve the goal of making human performance more uniform, if these were important causes of human failure. But, even from this viewpoint, a tool design focused on helping users with cases where they are least effective (or most vulnerable to the effects of fatigue, for instance) could still be the more effective solution (Strigini, Povyakalo et al. 2003). Even when designers lack information to identify these categories of cases, there may be some reward in building the tools on principles different from replicating human behaviour. The blind pursuit of diversity has proven effective, after all, in other areas of system design (Strigini 2005). However, for humans to accept hints from such tools, it may be more important that the tool provides convincing explanations of its behaviour, lest users become accustomed to ignoring hints that they cannot make sense of (*cf* the observation that CAD users feel the need to explain successes and errors of the CAD tool in terms of regular patterns of behaviour (Hartswood and Procter 2000; Hartswood, Procter et al. 2003)).

7.2 Reduce the effect of incorrect prompting on user error

The instructions for using the tool in our case study say that readers must not allow machine prompts (or lack thereof) to influence how they interpret features in the X-ray images. If readers comply, then incorrect prompting of a case could not produce a false negative, except on cases on which the reader failed to notice the symptoms to start with: computer support "could do no harm". But can users actually comply with these guidelines? Much of the skill that experts utilise is at the level of non-explicit pattern recognition rules and heuristics. If experts slowly adapted to relying on the tool's prompts for advice, at least for some types of cases, they may not realise that they are doing so. At least two forms of protection can be pursued:

- make it easier for the user to obey the prescribed procedure, or more difficult to violate it. For instance, the user interface could allow users to mark all features they are going to consider in their

decision, before showing the tool's prompts. However, such measures may be cumbersome to enforce, especially with high load on the users, causing them to take advantage of all available help;

- make the user more immune to being influenced in the wrong way by the tool. A form of protection is to remind users of the possibility of machine failure by giving frequent examples of incorrect prompting: in training, just as e.g. pilot simulator training includes an unrealistically high rate of mechanical failure, or even better, where socially and organisationally acceptable, in normal use.

Several authors assume, both for cancer screening CAD and for advisory systems in general, that the phenomenon of incorrect prompting causing user error is limited to users who lack experience with the tool, and disappears with use. This seems a one-sided over-optimistic statement, if not supported by decisive empirical evidence:

- experience will help users to recognise that the tool is fallible, but it will also teach them that, for instance, in many situations it is normally reliable. CAD tools are tuned to have low FN rate at the cost of a high FP rate, so they are indeed quite reliable for the kinds of cancer signs they target.
- experience will teach them to recognise some situations in which the tool tends to fail, but this will be more likely the more the situation is one in which the user is reliable and so needs the tool's support less.

So, the issue is not whether experience or special training or other factors will have some mitigating effect on user error rates, but the magnitude of these effects, when combined, and its sign and magnitude when combined with that of detrimental factors.

Another issue specific to our case study is whether it is right to insist that readers not be influenced by the tool not prompting a feature about which they were uncertain. After all, with highly sensitive alerting tools, absence of alerts is a good indicator that no alarm should be raised. As users appear to recognise and exploit this fact to reduce their workload, it might be appropriate to develop explicit guidelines about how to use this kind of indication.

8 Conclusions

We have discussed the outcomes of a case study with a form of computer aided decision making, trying to elicit implications of general interest, in particular in the area of possible detrimental effects of a computer aid.

One outcome is further evidence against the simplistic view that "computer support can do no harm". Some mechanisms that may make a computer aid harmful, like user complacency, are intuitively obvious. Others that we have hypothesised here are subtler. An important implication for designers is that they cannot ignore the possibility of such harmful effects; they should consider whether the overall effect of computer support can be made sufficiently positive. In cases of high uncertainty about these factors, they must consider whether effective precautions can be adopted in designing the whole decision system (not just the computer part of it), and how the effects of computer aids can be monitored in use.

Another known lesson that our findings support is that the problem in human-computer interaction goes way beyond designing the visible human-computer interfaces. Much codified knowledge about "human factors" is about the latter. But a computer may affect the dependability of the system of which it is part mostly through how it affects the mental processes of its users, quite independently of how friendly and "usable" it appears. In this case study, we would conjecture that design choices like the false positive-false negative trade-offs selected by designers for various categories of cases, and the way readers learn about the characteristics of the CAD tool, may be the determinant factors in the effects observed here.

From the methodological viewpoint, our experience confirms the usefulness of an interdisciplinary approach, exploiting the indications of diverse methods of investigation. Inspired by our experience modelling common-mode failure processes in diverse redundancy, we applied methods for exploratory data analysis that focused on the variation between behaviours of computers or people across the range of "cases". These methods revealed interesting patterns that would be missed by standard analyses

focused on the average effect of using a computer tool. We must underscore that these methods are a source of useful conjectures about the mechanisms of action of computer support, which can then be subjected to independent confirmation, but also provide indications of possible design problems even while this confirmation is missing.

Last but not least, recognising that the system of interest is the whole decision system, a composite human-machine, fault-tolerant system, rather than the computer part of it, is essential for a correct approach to its design.

Acknowledgements

The work described in this paper has been partly funded by the UK Engineering and Physical Sciences Research Council (EPSRC) through DIRC, the Dependability Interdisciplinary Research Collaboration. We thank: Mark Hartswood and our DIRC colleagues in Edinburgh and Lancaster Universities for the ethnographic work mentioned here and extensive discussions; R2 Technologies Inc. for their support in obtaining our test set; Paul Taylor and Jo Champness (from UCL) for granting us access to their data, facilitating the follow-up studies; all the mammogram readers who volunteered for our studies.

References

- Alberdi, E., A. A. Povyakalo, et al. (2004). "Effects of incorrect CAD output on human decision making in mammography." *Acad Radiol* **11**(8): 909-918.
- Alberdi, E., A. A. Povyakalo, et al. (2005). "The use of Computer Aided Detection tools in screening mammography: A multidisciplinary investigation." *Brit J Radiol*. **78**: 31-40.
- Astley, S. M. and F. J. Gilbert (2004). "Computer-aided detection in mammography." *Clin Radiol* **59**(5): 390-399.
- Azar, B. (1998). "Danger of automation: It makes us complacent." *APA monitor* **29**(7): 3.
- Bainbridge, L. (1983). "Ironies of automation." *Automation*: 775-779.
- Billings, C. E., J. K. Lauber, et al. (1976). NASA aviation safety reporting system. Moffet Field, U.S.A, NASA.
- Bisantz, A. M. and Y. Seong (2001). "Assessment of operator trust in and utilization of automated decision-aids under different framing conditions." *International Journal of Industrial Ergonomics* **28**(2): 85-97.
- BJR, S. I. (2005). *Computer-aided diagnosis*.
- Bliss, J. P. and S. A. Acton (2003). "Alarm mistrust in automobiles: how collision alarm reliability affects driving." *Applied Ergonomics* **34**(6): 499-509.
- Corcoran, D. W., J. L. Dennett, et al. (1972). "Cooperation of listener and computer in a recognition task: II. Effects of computer reliability and "dependent" versus "independent" conditions." *Journal of the Acoustical Society of America* **52**: 1613-1619.
- Cummings, M. L. (2004). *Automation bias in intelligent time critical decision support systems*. AIAA 1st Intelligent Systems Technical Conference, AIAA 2004, Chicago, IL.
- Dalal, N. P. and G. M. Kasper (1994). "The design of joint cognitive systems: The effect of cognitive coupling on performance." *International Journal of Human-Computer Studies* **40**: 677-702.
- Dassonville, I., D. Jolly, et al. (1996). "Trust between man and machine in a teleoperation system." *Reliability Engineering & System Safety* **53**(3): 319-325.
- de Vries, P., C. Midden, et al. (2003). "The effects of errors on system trust, self-confidence, and the allocation of control in route planning." *International Journal of Human-Computer Studies* **58**(6): 719-735.
- Dobson, A. J. (1990). *An introduction to generalized linear models*. London, Chapman and Hall.
- Dzindolet, M. T., S. A. Peterson, et al. (2003). "The role of trust in automation reliance." *International Journal of Human-Computer Studies* **58**(6): 697-718.
- Dzindolet, M. T., L. G. Pierce, et al. (2002). "The perceived utility of human and automated aids in a visual detection task." *Human Factors* **44**(1): 79-94.
- FDA (1998). Pre-market approval decision. Application P970058, US Food and Drug Administration.

- Fogg, B. J. and T. Hsiang (1999). The elements of computer credibility. CHI99, Pittsburgh, PA.
- Galletta, D. F., A. Durcikova, et al. (2005). "Does spell-checking software need a warning label?" Commun. ACM **48**(7): 82-86.
- Hartswood, M. and R. Procter (2000). "Computer-Aided Mammography: A Case Study of Error Management in a Skilled Decision-making Task." Journal of Topics in Health Information Management **20**(4): 38-54.
- Hartswood, M., R. Procter, et al. (2003). 'Repairing' the Machine: A Case Study of the Evaluation of Computer-Aided Detection Tools in Breast Screening. Eighth European Conference on Computer Supported Cooperative Work (ECSCW 2003), Helsinki, Finland.
- Hartswood, M., R. Procter, et al. (1998). Prompting in mammography: Computer-aided detection or computer-aided diagnosis? Proceedings of Medical Image Understanding and Analysis, MIUA 98, Leeds, UK.
- Hosmer, D. W., T. Hosmer, et al. (1997). "A comparison of goodness-of-fit tests for the logistic regression model." Stat in Med **16**(9): 965-980.
- Lee, J. D. and N. Moray (1994). "Trust, self-confidence, and operators' adaptation to automation." International Journal of Human-Computer Studies **40**: 153-184.
- Lee, J. D. and K. A. See (2003). "Trust in computer technology. Designing for appropriate reliance." Human Factors.
- Littlewood, B., P. Popov, et al. (2002). "Modelling software design diversity - a review." ACM Computing Surveys **33**(2): 177-208.
- Martin, C. D. (1993). "The myth of the awesome thinking machine." Commun ACM **36**(4): 39-44.
- Meyer, J. (2001). "Effects of warning validity and proximity on responses to warnings." Hum Factors **43**: 563-572.
- Meyer, J. (2004). "Conceptual issues in the study of dynamic hazard warnings." Human Factors **46**(2): 196-204.
- Meyer, J. and Y. Bitan (2002). "Why better operators receive worse warnings." Human Factors **44**(3): 343-353.
- Meyer, J., L. Feinsreiber, et al. (2003). Levels of automation in a simulated failure detection task. IEEE International Conference on Systems, Man and Cybernetics, Washington DC, USA.
- Meyer, J., L. Feinsreiber, et al. (2003). Levels of automation in a simulated failure detection task. IEEE International Conference on Systems, Man and Cybernetics, 2003, Washington DC, USA.
- Moray, N. (2003). "Monitoring, complacency, scepticism and eutactic behaviour." International Journal of Industrial Ergonomics **31**(3): 175-178.
- Mosier, K. L., L. J. Skitka, et al. (1998). "Automation bias: Decision making and performance in high-tech cockpits." International Journal of Aviation Psychology **8**: 47-63.
- Muir, B. M. (1987). "Trust between humans and machines, and the design of decision aids." International Journal of Man-Machine Studies **27**: 527-539.
- Muir, B. M. (1994). "Trust in automation: Part I. Theoretical issues in the study of trust and human intervention in automated systems." Ergonomics **37**: 1905-1922.
- Muir, B. M. and N. Moray (1996). "Trust in automation: Part II. Experimental studies of trust and human intervention in a process control simulation." Ergonomics **39**: 429-460.
- Parasuraman, R. (1987). "Human-computer monitoring." Human Factors **29**: 695-706.
- Parasuraman, R. and C. A. Miller (2004). "Trust and etiquette in high-criticality automated systems." Communications of the ACM **47**(4): 51-55.
- Parasuraman, R. and V. Riley (1997). "Humans and automation: Use, misuse, disuse, abuse." Hum Factors **39**: 230-253.
- Parasuraman, R. and V. Riley (1997). "Humans and automation: Use, misuse, disuse, abuse." Human Factors **39**: 230-253.
- Parasuraman, R., T. B. Sheridan, et al. (2000). "A model for types and levels of human interaction with automation." IEEE Transaction on systems, Man, & Cybernetics **30**(3): 286-297.
- Povyakalo, A. A., E. Alberdi, et al. (2004). Evaluating 'Human + Advisory computer' system: A case study. HCI2004, 18th British HCI Group Annual Conf., Leeds, British HCI Group.
- Povyakalo, A. A., E. Alberdi, et al. (2005). Divergent effects of computer prompting on the sensitivity of mammogram readers. CSR, City University, Technical Report.

- Singh, I. L., R. Molloy, et al. (1993). "Automation-induced "complacency": development of the complacency-potential rating scale." International Journal of Aviation Psychology **3**: 111-122.
- Singh, I. L., R. Molloy, et al. (1997). "Automation-induced monitoring inefficiency: role of display location." International Journal of Human-Computer Studies **46**(1): 17-30.
- Skitka, L. J. (2000). "Accountability and automation bias." Int J Hum-Comp Stud **52**: 701-717.
- Skitka, L. J., K. Mosier, et al. (1999). "Does automation bias decision making?" International Journal of Human-Computer Studies **51**(5): 991-1006.
- Skitka, L. J., K. Mosier, et al. (2000). "Accountability and automation bias." International Journal of Human-Computer Studies **52**(4): 701-717.
- Sorkin, R. D. and D. D. Woods (1985). "Systems with human monitors: A signal detection analysis." Human-Computer Interaction **1**: 49-75.
- Strigini, L. (2005). Fault Tolerance Against Design Faults. Dependable Computing Systems: Paradigms, Performance Issues, and Applications. H. Diab and A. Zomaya, J. Wiley & Sons.
- Strigini, L., A. A. Povyakalo, et al. (2003). Human-machine diversity in the use of computerised advisory systems: a case study. 2003 Int. Conf. on Dependable Systems and Networks (DSN'03), San Francisco, IEEE.
- Strigini, L., A. A. Povyakalo, et al. (2003). Human-machine diversity in the use of computerised advisory systems: a case study. DSN 2003, International Conference on Dependable Systems and Networks, San Francisco, U.S.A.
- Tan, G. and S. Lewandowsky (1996). A comparison of operator trust in humans versus machines. Presentation of First International Cyberspace Conference on Ergonomics.
- Taylor, P., J. Champness, et al. (2005). "Impact of computer-aided detection prompts on the sensitivity and specificity of screening mammography." Health Technology Assessment **9**(6).
- Taylor, P., R. Given-Wilson, et al. (2004). "Assessing the impact of CAD on the sensitivity and specificity of film readers." Clin Radiol. 2004 Dec;**59**(12):1099-105. **59**(12): 1099-1105.
- Thackray, R. I. and R. M. Touchstone (1989). "Detection efficiency on an air traffic control monitoring tasks with and without computer aiding." Aviation, Space, and Environmental Medicine **60**: 744-748.
- Tseng, S. and B. J. Fogg (1999). "Credibility and computing technology." Communications of the ACM **42**(5): 39-44.
- Wickens, C. D. and S. R. Dixon (2005). Is there a Magic Number 7 (to the Minus 1)? The Benefits of Imperfect Diagnostic Automation: A Synthesis of the Literature. Savoy, Illinois, University of Illinois at Urbana-Champaign: 1-11.
- Wiener, E. L. (1981). Complacency: is the term useful for air safety? 26th Corporate Aviation Safety Seminar, Denver, CO, Flight Safety Foundation, Inc.
- Zheng, B., M. A. Ganott, et al. (2001). "Soft-Copy Mammographic Readings with Different Computer-assisted Detection Cuing Environments: Preliminary Findings." Radiology **221**(3): 633-640.

Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification

Barboni Eric¹, Conversy Stéphane^{1,2}, Navarre David¹ & Palanque Philippe¹

¹ LIHS – IRIT, Université Paul Sabatier
118 route de Narbonne, 31062 Toulouse Cedex 4
{barboni, conversy, navarre, palanque}@irit.fr
<http://lihs.irit.fr/{barboni,navarre,palanque}>

² ENAC – DTI/SDER – Ecole Nationale de l'Aviation Civile
7, avenue Edouard Belin, 31055 Toulouse.
conversy@enac.fr

Abstract. The purpose of ARINC 661 specification [1] is to define interfaces to a Cockpit Display System (CDS) used in any types of aircraft installations. ARINC 661 provides precise information for communication protocol between application (called User Applications) and user interface components (called widgets) as well as precise information about the widgets themselves. However, in ARINC 661, no information is given about the behaviour of these widgets and about the behaviour of an application made up of a set of such widgets. This paper presents the results of the application of a formal description technique to the various elements of ARINC 661 specification within an industrial project. This formal description technique called Interactive Cooperative Objects defines in a precise and non-ambiguous way all the elements of ARINC 661 specification. The application of the formal description techniques is shown on an interactive application called MPIA (Multi Purpose Interactive Application). Within this application, we present how ICO are used for describing interactive widgets, User Applications and User Interface servers (in charge of interaction techniques). The emphasis is put on the model-based management of the feel of the applications allowing rapid prototyping of the external presentation and the interaction techniques. Lastly, we present the CASE (Computer Aided Software Engineering) tool supporting the formal description technique and its new extensions in order to deal with large scale applications as the ones targeted at by ARINC 661 specification.

1 Introduction

Interactive applications embedded in cockpits are the current trend of evolution promoted by several aircraft manufacturer both in the field of civil and military systems [7, 10]. Embedding interactive application in civil and military cockpit is expected to provide significant benefits to the pilots by providing them with easier to use and more efficient applications increasing the communication bandwidth between pilots and systems. However, this technological enhancement comes along with several problems that have to be taken into account with appropriate precautions.

ARINC specification 661 (see next section), aims at providing a common ground for building interactive applications in the field of aeronautical industry. However, this standard only deals with part of the issues raised. The aim of this paper is to propose a formal description technique to be used as a complement to ARINC 661 for the specification, design, implementation and validation of interactive application.

The paper is structured as follows. Next section introduces ARINC 661 specification to define software interfaces for a Cockpit Display System. It presents informally the content of the specification but also its associated architecture that has to be followed in order to build ARINC-661-compliant interactive applications. Section 3 presents the ICO formalism, a formal description technique for the design of safety critical interactive applications. This description technique has already been applied in various domains including Air Traffic Control applications, multimodal military cockpits or multimodal satellite ground segments. Its applicability to cockpit display system and its compatibility with ARINC specification 661 is discussed and extensions that had to be added are also presented in section 4. Section 5 presents the use of the formal description technique on an interactive application called MPIA (Multi Purpose Interactive Application) currently available in some cockpits of regional aircrafts. Last section of the paper deals with conclusions and perspectives to this work.

2 ARINC 661 specification

This section presents, in an informal way, the basic principles of ARINC 661 specification. The purpose of this section is to provide a description of the underlying mechanisms of ARINC 661 specification and more precisely how its content influences the behaviour and the software architecture of interactive applications embedded in interactive cockpits.

2.1 Purpose and Scope

The purpose of ARINC 661 specification (ARINC 661, 2002) is to define interfaces to a Cockpit Display System (CDS) used in interactive cockpits that are now under deployment by several aircraft manufacturers including Airbus, Boeing and Dassault. The CDS provides graphical and interactive services to user applications (UA) within the flight deck environment. Basically, the interactive applications will be executed on Display Units (DU) and interaction with the pilots will take place through the use of Keyboard and graphical input devices like the Keyboard Cursor Control Unit (KCCU).

ARINC 661 dissociates, on one side, input and output devices (provided by avionics equipment manufacturers) and on the other side the user applications (designed by aircraft manufacturers). Consistency between these two parts is maintained through a communication protocol:

- Transmission of data to the CDS, which can be displayed to the flight deck crew.
- Reception of input (as events) from interactive items managed by the CDS.

In the field of interactive systems engineering, interactive software architectures such as Seeheim [14] or Arch [9] promote a separation of the interactive system in at least three components: presentation part (in charge of presenting information to and receiving input from the users), dialogue part (in charge of the behaviour of the system i.e. describing the available interface elements according to the current state of the application) and functional core (in charge of the non interactive functions of the system). The CDS part may be seen as the presentation part of the whole system, provided to crew members, and the set of UAs may be seen as the merge of both the dialogue and the functional core of this system.

2.2 User Interface Components in ARINC 661

The communication between the CDS and UAs is based on the identification of user interface components hereafter called widgets. ARINC 661 defines a set of 42 widgets that belong to 6 categories. Widgets may be any combination of “container”, “graphical representation” of one or more data, “text string” representations, “interactive”, dedicated to “map management” or may “dynamically move”.

In ARINC 661, each widget is defined by:

- a set of states classified in four levels (visibility, inner state, ability, visual representation),
- a description in six parts (definition section, parameters table, creation structure table, event structure table, run-time modifiable parameter table, specific sections).

The main drawback of this description is the lack of description of the behaviour itself. Even if states are partially described, dynamic aspects such as state changes are informally described. As stated in ARINC 661 (section 1.0 introduction), the main paradigm is here based on this comment:

“A UA should not have any direct access to the visual representations. Therefore, visual presentations do not have to be defined within the ARINC 661 interface protocol. Only the ARINC 661 parameter effects on graphical representation should be described in the ARINC 661 interface. The style guide defined by the OEM should describe the “look and feel” and thus, provide necessary information to UAs for their HMI interface design.”

An additional textual description called SRS (for Software Requirement Specification), informally defines the look and feel of a CDS (Cockpit Display System). This SRS is designed by each manufacturer of airline electronic equipment (we worked with a draft document provided by Thales Avionics). This kind of document describes both the appearance and the detailed expected behaviour of each graphical or interactive component.

2.3 Overview of our contribution to ARINC 661

One of the goals of the work presented in this paper is to define an architecture that clearly identifies each part of this architecture and their communication, as shown on Fig 1. The aim of this architecture is also to clearly identify which components will be taken into account in the modelling process and which ones are taken into account in a different way by exploiting SVG facilities. The architecture has two main advantages:

1. Every component that has an inner behaviour (server, widgets, UA, and the connection between UA and widgets, e.g. the rendering and activation functions) is fully modelled using the ICO formal description technique.
2. The rendering part is delegated to a dedicated language and tool (such as SVG).

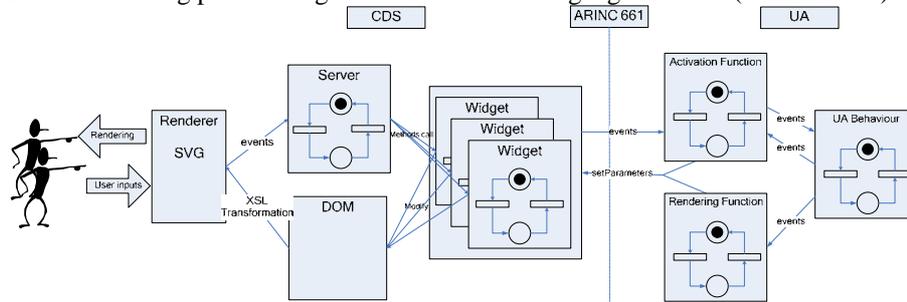


Fig 1. Detailed architecture to support ARINC 661 specification

The following section recalls the basics of ICO notation and presents a new extension that has been required in order to be able to address all the modelling challenges put forward by interactive cockpit applications compliant with ARINC 661 specification, and then present the connection to SVG. Lastly, a real case study illustrates this architecture and how modelling all the elements of ARINC 661 specification are addressed using ICOs formal description technique.

3 ICO modelling of ARINC 661 components

We use the ICO formalism to describe formally the behaviour of the ARINC components. This section first briefly recalls the main features of the ICO formalism. We encourage the interested reader to look at [13, 11] for a complete presentation of the formal description technique and the environment supporting it. The second part is dedicated to the extensions that had to be defined in order to address the specificities of interactive applications compliant with ARINC 661 specifications.

3.1 Overview of the ICO formalism

The Interactive Cooperative Objects (ICOs) formalism is a formal description technique dedicated to the specification of interactive systems [4, 11]. It uses concepts borrowed from the object-oriented approach to describe the structural or static aspects of systems, and uses high-level Petri nets [8] to describe their dynamic or behavioural aspects. ICOs are dedicated to the modelling and the implementation of event-driven interfaces, using several communicating objects to model the system, where both behaviour of objects and communication protocol between objects are described by Petri nets. The formalism made up of both the description technique for the communicating objects and the communication protocol is called the Cooperative Objects formalism (CO).

ICOs are used to provide a formal description of the dynamic behaviour of an interactive application. An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e., how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e., when and how the application displays information relevant to the user). Time-out transitions are special transitions that do not belong to the categories above.

An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented [12]. The specification can also be validated using analysis and proof tools developed within the Petri nets community and extended in order to take into account the specificities of the Petri net dialect used in the ICO formal description technique.

3.2 ICO improvements

Two main issues have been raised while working with ARINC 661 specification that have not been encountered in previous work we have done in the field of interactive systems' specification and modeling.

- The first one is related to the management of rendering information in a more independent and structured way in order to be able to dissociate as much as possible the graphical appearance of interactive components from their behavior. This is one of the basics of interactive cockpit applications compliant with ARINC 661 specification as (as stated above) these two sides of the interactive cockpit applications are described in two different documents (communication protocol and abstract behavior in ARINC 661 specification while presentation and detailed behavior are described in the SRS (System Requirement Specifications)).
- The second one is related to the fact that ARINC 661 specification does not exploit current windows manager available in the operating system (as this is the case for Microsoft Windows applications for instance). On the opposite, the manufacturer in charge of developing the entire ARINC 661 architecture is also in charge of developing all the components in charge of the management of input devices, device drivers and to manage the

graphical structure of the interactive widgets. In order to handle those aspects we have defined a denotational semantics (in terms of High-level Petri nets) of both the rendering and the activation functions. Beforehand, these functions were only partly defined (relying on the underlying mechanisms provided by the window manager) and implemented using a particular java API thus making much more limited the verification aspects of these aspects of the specification. Indeed, the work presented here addresses at the same level of formality, applications, widgets and user interface server (also called window manager). Besides, the connections and communications between these three parts are also formally described.

Next section presents in details the various mechanisms that have been defined in order to handle the low level management of input devices and focuses on one specific aspect called picking which correspond to the window manager activating of finding the interactive component that was the target of the user when an event has been produced. The case study in section 4 shows on a concrete example how those elements are combined for describing User Applications, Widgets and User Interface servers.

4 MPIA case study

MPIA is a User Application (UA) that aims at handling several flight parameters. It is made up of 3 pages (called WXR, GCAS and AIRCOND) between which a crew member is allowed to navigate using 3 buttons (as shown by Fig 2). WXR page is in charge managing weather radar information; GCAS is in charge of the Ground Anti Collision System parameters while AIRCOND deals with settings of the air conditioning.



Fig 2. Snapshots of the 3 pages of the UA MPIA

In this section, we present the modelling of a simple widget and its link to SVG rendering, then we briefly present the classical modelling of a user application to show the extension made to ICOs, and finally we present parts of the server. The purpose is not here to present the whole specification which is made up of about 40 models, but only to present brief extracts to show all bricks of the modelling.

4.1 Modelling ARINC 661 interactive widgets

The whole modelling process of ARINC 661 interactive components using ICO is fully described in [12]. The additional feature consists in using the rendering process described above, based on replacing the classical code-based rendering methods with rendering methods that modify the SVG Document Object Model. Rendering is the process of transforming a logical description (conceptual model) of an interactive component to a graphical representation (perceptual model). In previous similar works, we specified rendering with Java code, using the Java2D API. However, describing graphics with an imperative language is not an easy task, especially when one tries to match a particular look. Furthermore, the java code for graphics is embedded into the model, which makes it hard to change for another look. This is even more difficult when several components share a common part of the graphical representation, for instance when components must have a similar style and when this style has to be changed.

To overcome these two problems, we changed for an architecture that uses declarative descriptions of the graphical part and that supports transformations from conceptual models to graphical representations. These two elements exploit XML-based languages from the W3C: the SVG language for graphical representation, and the XSLT language for transformation. SVG is an xml-based vector graphics format: it describes graphical primitives in terms of analytical shapes and transformations. XSLT is an xml-based format that describes how to transform an xml description (the source) to another xml description (the target). An XSLT description is called a “stylesheet”. Due to space constraints this work is not presented in the next section as we focus on the behavioural aspects of models.

4.2 Modelling User Applications

Modelling a user application using ICO is quite simple as ICO has already been used to model such kind of interactive applications. Indeed, UAs in the area of interactive cockpits correspond to classical WIMP interfaces,

As the detailed specification is not necessary to expose the modification of ICO, we only present an excerpt of the models that have been produced to build the MPIA application. This excerpt is the first page (WXR) of the application (left part of Fig 2).

4.2.1 Behaviour

Fig 3 shows the entire behaviour of page WXR which is made up of two non connected parts:

- The upper part aims at handling events from the 5 CheckButtons and the modification implied of the MODE_SELECTION that might be one of five possibilities (OFF, STDBY, TST, WXON, WXA). Value changes of token stored in place *Mode-Selection* are described in the transitions while

variables on the incoming and outgoing arcs play the role of formal parameters of the transitions.

- The lower part concerns the handling of events from the 2 PicturePushButton and the EditTextNumeric. Interacting with these buttons will change the state of the application.

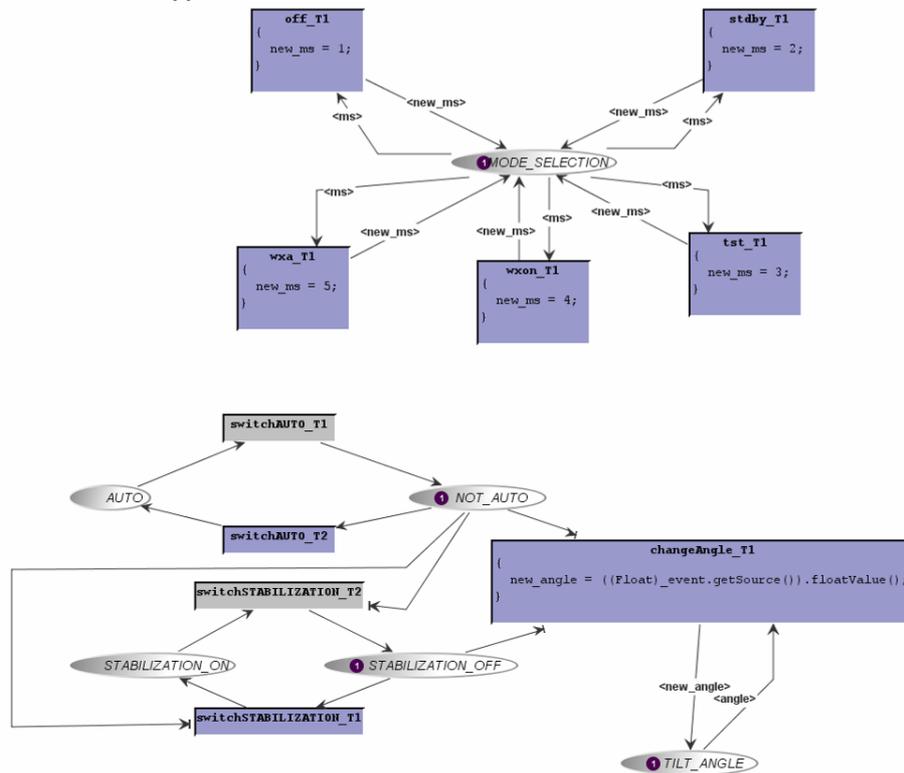


Fig 3. Behaviour of the page WXR

4.2.2 Activation function

Fig 4 shows an excerpt of the activation function for page WXR.

	Widget	Event	UserService	ActivationRendering
wxrOFFAdapter	off_CheckButton	A661_INNER_STATE_SELECT	off	setWXRModeSelectEnabled
wxrSTDBYAdapter	stdby_CheckButton	A661_INNER_STATE_SELECT	stdby	setWXRModeSelectEnabled
wxrTSTAdapter	tst_CheckButton	A661_INNER_STATE_SELECT	tst	setWXRModeSelectEnabled
wxrWXONAdapter	wxon_CheckButton	A661_INNER_STATE_SELECT	wxon	setWXRModeSelectEnabled
wxrWXAAdapter	wxa_CheckButton	A661_INNER_STATE_SELECT	wxa	setWXRModeSelectEnabled
autoAdapter	auto_PicturePushButton	A661_EVT_SELECTION	switchAUTO	setWXRtiltSelectionEnabled
stabAdapter	stab_PicturePushButton	A661_EVT_SELECTION	switchSTABILIZATION	setWXRtiltSelectionEnabled
tiltAngleAdapter	tiltAngle_EditBox	A661_STRING_CHANGE	changeAngle	setWXRtiltSelectionEnabled

Fig 4. Activation Function of the page WXR

From this textual description, we can derive the ICO model shown on Fig 5. The left part of this figure presents the full activation function, which is made up of as many sub Petri nets as there are lines in the textual activation function. The upper right hand side of the figure emphasises on of these sub Petri nets. It describes how the availability of the associated widget is modified according to some changes in the WXR behaviour. The lower right hand part of the Figure shows the general pattern associated to one line of the activation function: It describes the handling of the event raised par the corresponding widget, and how it is linked to an event handler in the WXR behaviour.

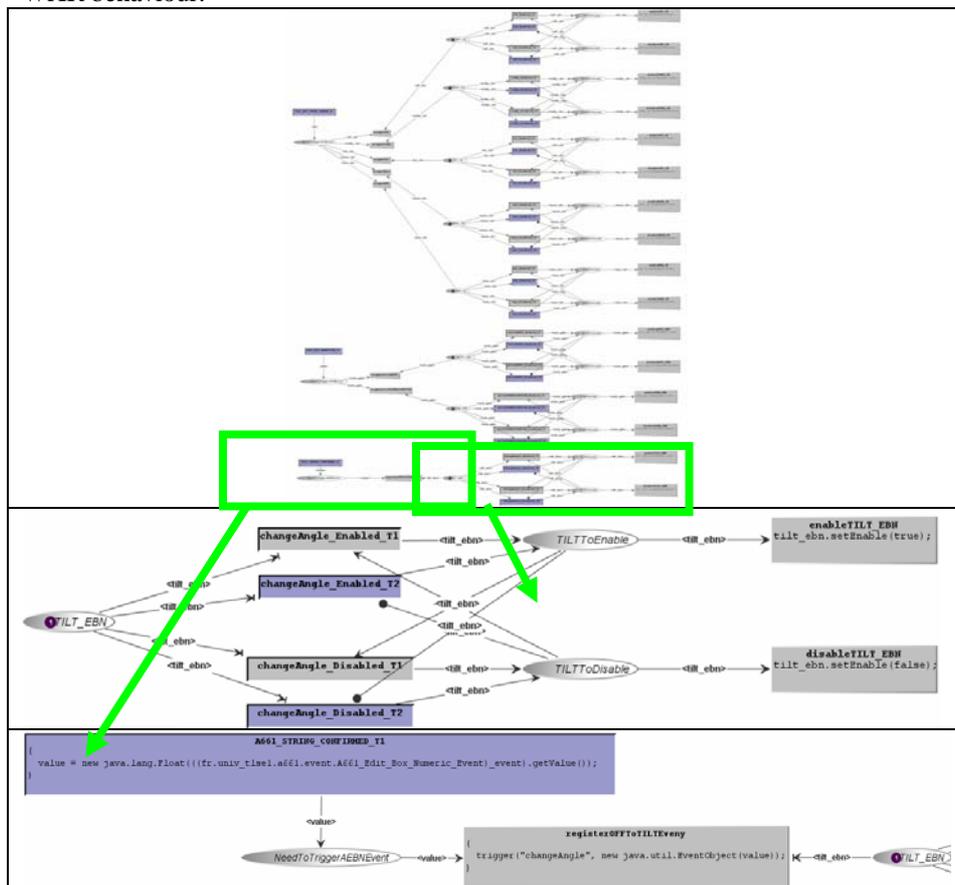


Fig 5. Activation Function of the page WXR expressed in Petri nets

The use of Petri nets to model the activation function is made possible thanks to the event communication available in the ICO formalism. As this kind of communication is out of the scope of this paper, we do not present the models responsible in the registration of events-handlers needed to allow the communication between behaviour, activation function and widgets. More information about this mechanism can be found in [2].

4.2.3 Rendering Function

The modelling of the rendering function (shown on Fig 6) into Petri nets (shown on Fig 7) works the same way as for the activation function, i.e. for each line in the rendering function, there is a pattern to express that in Petri nets. This is why we do not detail more the translation.

	ObCSNode name	ObCS event	Rendering method
modeSelectionAdapter	MODE_SELECTION	token_enter <int m>	showModeSelection(m)
tiltAngleAdapter	TILT_ANGLE	token_enter <float a>	showTiltAngle(a)
initAutoAdapter	AUTO	marking_reset	showAuto(true)
autoAdapter	AUTO	token_enter	showAuto(true)
notAutoAdapter	AUTO	token_remove	showAuto(false)
initStabAdapter	STABILIZATION_ON	marking_reset	showStab(true)
stabAdapter	STABILIZATION_ON	token_enter	showStab(true)
notStabAdapter	STABILIZATION_ON	token_remove	showStab(false)

Fig 6. Rendering Function of the page WXR

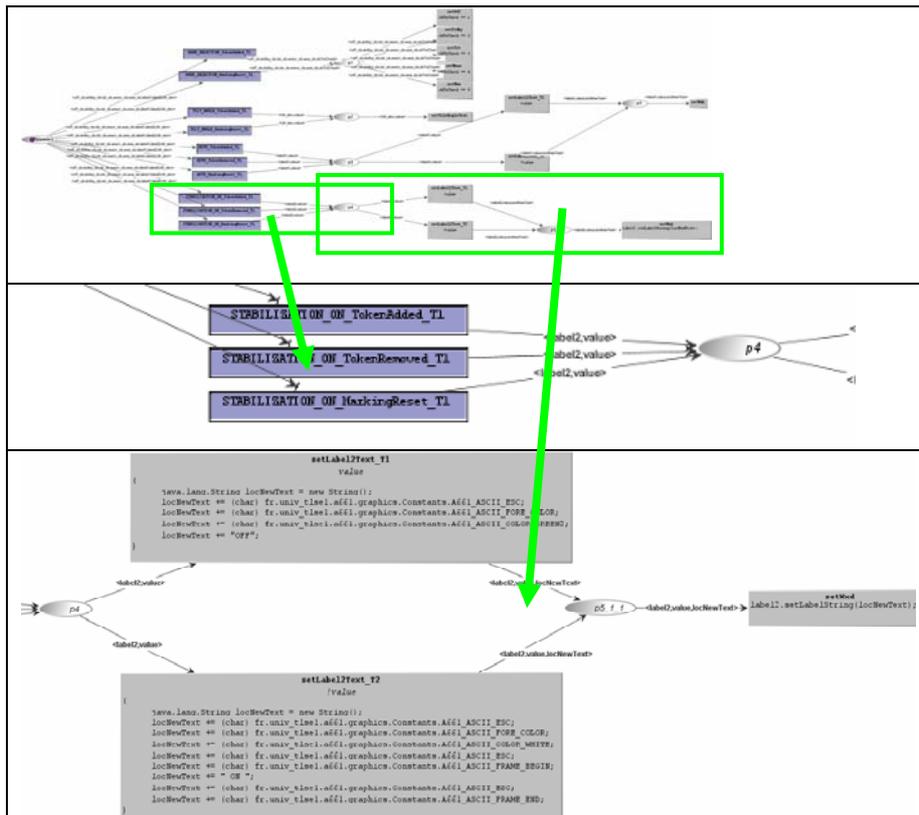


Fig 7. Rendering Function of the page WXR expressed in Petri nets

4.3 Modelling User Interface Server

The user interface server manages the set of widgets and the hierarchy of widgets used in the User Applications. More precisely, the user interface server is responsible in handling:

- The creation of widgets
- The graphical cursors of both the pilot and his co-pilot
- The edition mode
- The mouse and keyboard events and dispatching it to the corresponding widgets
- The highlight and the focus mechanisms
- ...

As it handles many functionalities, the complete model of the sub-server (dedicated in handling widgets involved in the MPIA User Application) is complex and difficult to manipulate without an appropriate tool. As the detailed model is out of the scope of this paper, Fig 8 only present an overview of the complete model.

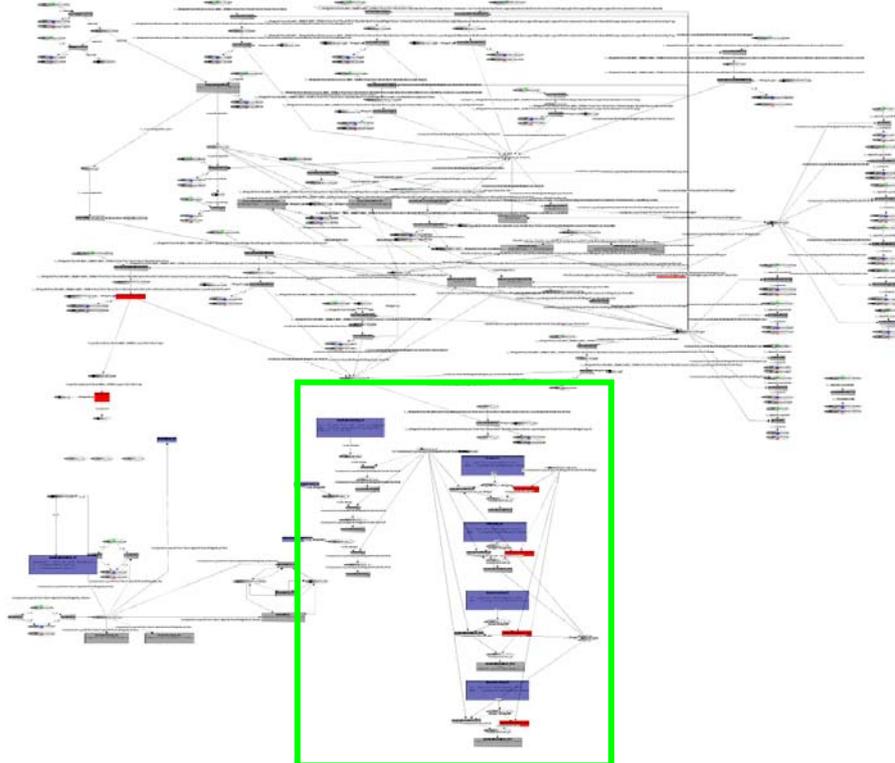


Fig 8. Overview of the complete model of the user interface server.

The rectangle at the bottom of Fig 8 represents the part of the model of the server in charge of the interaction technique and input devices management. The rest of the model corresponds to the management of the widgets.

4.4 Modelling the complete MPIA User Application

We do not present here the full model of the user application MPIA neither the one of the user interface server, but the formal description technique ICO has been used to model in a complete and non ambiguous way all the pages and the navigation between pages for such user application, and still produces low-sized and readable models. Modelling Activation functions and Rendering functions using Petri nets, legitimates the use of the table notation as a readable way to express the connection between the dialog and the presentation parts.

Another issue is that the models of the user application MPIA can both be connected to the modelled CDS or to an implemented CDS, using a special API, as it respects the ARINC 661 specification. As testing an implemented user application is still a problem that has to be solved, especially when the UA is connected to a real CDS, a model based approach may support testing at different levels:

1. Test a modelled user application on the modelled CDS.
2. Test the modelled user application on the CDS implemented by the manufacturer.
3. Code and test the user application on the implemented CDS.

The first step promotes a very iterative prototyping process where both the User Application and the CDS may be modified, as the second step allows user testing on the real interactive system (CDS), with classical prototyping facilities provided by the models expressed in ICO of the User Application.

The MPIA application has been fully modelled and can be executed on the CDS modelled using the ICO formalism. However, it has also been connected on a CDS developed on an experimental test bench as shown in Fig. 9.



Fig. 9. The MPIA application modelled using ICO connected to experimental CDS at THALES

5 Conclusions and Perspectives

This paper has presented the use of a formal description technique for describing interactive components in ARINC specification 661. Beyond that, we have shown that this formal description technique is also adequate for interactive applications embedding such interactive components. One of the advantages of using the ICO formal description technique is that it provides additional benefits with respect to other notations such as statecharts as proposed in [15]. Thanks to its Petri nets basis the ICO notations makes it possible to model behaviours featuring an infinite number of states (as states are modelled by a distribution of tokens in the places of the Petri nets). Another advantage of ICOs is that they allow designers to use verification techniques at design time as this has been presented in [3]. These verification techniques are of great help for certification purposes.

We are currently developing techniques for providing support to certification processes by allowing verification of compatibility between the behavioural description of the interactive application and task model describing nominal or unexpected pilots behaviour. Support is also provided through the verification of interactive system safety and liveness properties such as the fact that whatever state the system is in there is always at least one interactive element available.

6 Acknowledgements

The work presented in the paper is partly funded by DPAC (Direction des Programmes de l'Aviation Civile) under contract #00.70.624.00.470.75.96. Special thanks are due to our colleagues at THALES P. Cazaux and S. Marchal.

7 References

1. ARINC 661 specification: Cockpit Display System Interfaces To User Systems, Prepared by Airlines Electronic Engineering Committee, Published by AERONAUTICAL RADIO, INC, april 22, 2002.
2. Bastide R., Navarre D., Palanque P., Schyn A. & Dragicevic P. A Model-Based Approach for Real-Time Embedded Multimodal Systems in Military Aircrafts. Sixth International Conference on Multimodal Interfaces (ICMI'04) October 14-15, 2004 Pennsylvania State University, USA.
3. Bastide R., David Navarre & Philippe Palanque. Tool Support for Interactive Prototyping of Safety Critical Interactive Applications. In Encyclopedia of HCI, C. Gaoui (Ed.). ISBN: 1-59140-562-9. Hard Cover. Publisher: Idea Group Reference Pub Date: July 2005. Pages: 650.
4. Bastide R., Ph. Palanque A Petri Net Based Environment for the Design of Event-Driven Interfaces. 16th International Conference on Application and theory of Petri Nets (ATPN'95), LNCS, Springer Verlag, Torino, Italy, 20-22 June 1995.

5. Beaudoux O., 2005. XML Active Transformation (eXAcT): Transforming Documents within Interactive Systems. Proceedings of the 2005 ACM Symposium on Document Engineering (DocEng 2005), ACM Press, pages 146-148.
6. Blanch R., Michel Beaudouin-Lafon, Stéphane Conversy, Yannick Jestin, Thomas Baudel and Yun Peng Zhao. INDIGO : une architecture pour la conception d'applications graphiques interactives distribuées. In Proceedings of IHM 2005, pages 139-146, Toulouse - France, September 2005.
7. Faerber R. Vogl T. & Hartley D. Advanced Graphical User Interface for Next Generation Flight Management Systems. In proceedings of HCI Aero 2000, pp. 107-112.
8. Genrich H. J.. (1991). Predicate/Transition Nets, in K. Jensen and G. Rozenberg (Eds.), High-Level Petri Nets: Theory and Application. Springer Verlag, Berlin, pp. 3-43.
9. Gram C., Cockton G. (Editors). Design principles for interactive software. Chapman et Hall ed.1995.
10. Marrenbach J., Kraiss K-F. Advanced Flight Management System: A New Design and Evaluation Results. In proceedings of HCI Aero 2000, pp. 101-106.
11. Navarre D., Palanque, Philippe, Bastide, Rémi. A Tool-Supported Design Framework for Safety Critical Interactive Systems in Interacting with computers, Elsevier, Vol. 15/3, pp 309-328, 2003.
12. Navarre D., Philippe Palanque & Rémi Bastide. A Formal Description Technique for the Behavioural Description of Interactive Applications Compliant with ARINC 661 Specification. HCI-Aero'04 Toulouse, France, 29 September-1st October 2004.
13. Palanque P., R. Bastide. Petri nets with objects for specification, design and validation of user-driven interfaces. In proceedings of the third IFIP TC 13 conference on Human-Computer Interaction, Interact'90. Cambridge 27-31 August 1990 (UK).
14. Pfaff, G. E. (Hrsg.): User Interface Management Systems, Proceedings, Workshop on User Interface Management Systems, Seeheim,(1. - 3.11.1983); Springer Verlag 1983.
15. Sherry L., Polson P., Feary M. & Palmer E. When Does the MCDU Interface Work Well? Lessons Learned for the Design of New Flightdeck User-Interface. In proceedings of HCI Aero 2002, AAAI Press, pp. 180-186.
16. Souchon, N., Vanderdonck, J., A Review of XML-Compliant User Interface Description Languages, Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003), Jorge, J., Nunes, N.J., Falcao e Cunha, J. (Eds.), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391.
17. UsiXML, <http://www.usixml.org/?view=news>.
18. Villard, L. and Layaïda, N. 2002. An incremental XSLT transformation processor for XML document manipulation. In Proceedings of the 11th international Conference on World Wide Web (Honolulu, Hawaii, USA, May 07 - 11, 2002). WWW '02. ACM Press, New York, NY, 474-485.

Exploitation of Formal Specification Techniques for ARINC 661 Interactive Cockpit Applications

Eric Barboni, David Navarre, Philippe Palanque & Sandra Basnyat

LIHS-IRIT, Université Paul Sabatier (Toulouse 3),
118, route de Narbonne, 31062 Toulouse Cedex 4, France
{barboni, navarre, palanque, basnyat}@irit.fr

ABSTRACT

In the field of safety-critical interactive systems, the use of a formal specification technique is extremely valuable because it provides non-ambiguous, complete and concise ways of describing the behaviour of the systems. The advantages of using such formalisms are widened if they are provided with formal analysis techniques that allow proving properties about the models, thus giving an early verification to the designer before the application is actually implemented. Usually, one of the points put forward while designing such a formal description technique is to provide a notation powerful enough for describing/specifying the systems under consideration. In previous papers [14, 17] we have shown how the *expressive power* of the Interactive Cooperative Objects (ICO) formalism is sufficient for describing interactive applications of interactive cockpits or ground segments for satellite control rooms for instance. This paper addresses the issues related to the *applicability power* of the formalism and describes the extensions that have been defined and implemented to make it usable for real size interactive applications and even in an industrial context. The application domain we report on is the ARINC 661 specification standard for interactive cockpit applications. We first present the challenges raised by these kinds of applications and the basic principles of the standard. We then detail how we adapted the ICO formalism (and its CASE tool Petshop) with three innovative techniques, to specifically deal with such large and real-life safety-critical interactive systems. The solutions include models restructuring and visualization techniques (label hiding, virtual places and mini-view). We present these solutions using a case study showing all the aspects of the applications compliant with ARINC 661 specification, namely Server, User Applications and Widgets specifications.

KEYWORDS

ARINC 661 specification, formal description techniques, interactive software engineering, Interactive Cockpits.

INTRODUCTION

In the domain of the design and construction of safety-critical interactive systems, the use of a formal specification technique is extremely valuable because it provides non-ambiguous, complete and concise notations. The advantages of using such formalisms are widened if they are provided

with formal analysis techniques that allow proving properties about the design, thus giving an early verification to the designer before the application is actually implemented [7, 8 or 19].

The complete specification of interactive applications is now increasingly considered as a requirement in the field of software for safety critical systems. This is due to the fact that interactive systems are used as the main control interface. As the user interface part of command and control systems may represent a huge quantity of code, user interface design and construction tools must provide ways to address this complexity as well as the reliability. This paper does not address user interface design issues as it focuses on the notation and tools for their specification and construction. Support dealing with code management only (as typically provided by software development tools), is not enough and there is thus a critical need for addressing this complexity at a higher level of abstraction than software code. This paper argues that one possible way to deal with these issues is to follow the same path as in the field of software engineering where modeling activities and model-based approaches (also called model-driven approaches) take the lead with standards like UML [6]. Besides, for this kind of system (as argued in [15]) safety and usability aspects cannot be considered independently and development processes must be able to address them in a balanced and consistent way. While these concerns are typical of the development process of any type of software systems, safety critical ones put a special emphasis on the certification phase. Indeed, developers of such systems are in charge of demonstrating that the system is 'acceptably safe' before certification authorities grant regulatory approval. It is widely recognised that software qualification and system certification costs dwarf all other aspects of developing, installing and updating a Cockpit Display System (CDS) as argued in [2] p. 2.

While elements of solutions are available in the literature, scalability of the approaches to deal with real-life and real-size applications can often prove difficult due to the size and the number of models that have to be constructed and managed. In [14], we discussed the modelling of the ARINC 661 widgets and User Applications (UA) using the Interactive Cooperative Objects (ICO) formalism (in terms of functionality further presented in the ICO Formalism section). More particularly, in [14], we presented a small

User Application including only a very limited subset of the ARINC 661 widgets. We did not address the scalability capability of the formalism and its case tool. For example, the behaviour of a very simple widget (PicturePushButton that behaves as a standard command button) resulted in a not so simple model comprising of 47 places and 29 transitions, representing 16 reachable states. When it comes to more complicated widgets (like EditBoxes for instance) or to the modelling of the behaviour of the CDS (Cockpit Display System) server, the size of models is significantly larger and thus even more difficult to manage.

In the current paper we go beyond the previous paper by focusing on the “*applicability power*” of the ICO formalism. This aspect aims at improving the modelling process making the process of going from a given problem to a resulting model (representing a solution to that problem) easier. Modifications have been made at two different levels: firstly new model structuring mechanisms for handling complexity and secondly, new interaction and visualization techniques for editing and modifying models. These extensions are generic in the sense that they can be applied to other formal description techniques. We have applied the new structuring mechanism to a real-size application for interactive cockpits and present its improvement in terms of size and structure of models with respect to a more classical structuring technique. As for interaction and visualization techniques, we have implemented all of them in Petshop (the edition and simulation environment for ICO models) and we present how the modifications have an impact on the modelling process itself.

The paper is structured as follows. The following section introduces the ARINC 661 specification standard in an informal way, including the purpose and scope of this standard. Following this, we briefly present the case study that we use for exemplifying the issues raised and the solutions we propose for modelling interactive cockpit applications. We then provide an informal presentation of the ICO formalism and its CASE (Computer Aided Software Engineering) tool, Petshop. We discuss the limitations of the ICO formalism and the problems we encountered in our previous research work, and lastly present the modified model structuring technique and three extensions made to Petshop to cope with these scalability issues. The last section of the paper deals with conclusions and perspectives to this work.

AN OVERVIEW OF ARINC 661 SPECIFICATION

The Airlines Electronic Engineering Committee (AEEC) (an international body of airline representatives leading the development of avionics architectures) formed the ARINC 661 Working Group to define the software interfaces to the Cockpit Display System (CDS) used in all types of aircraft installations. The standard is called ARINC 661 - Cockpit Display System Interfaces to User Systems [1, 2].

The CDS (the software system embedded in an aircraft) provides graphical and interactive services to user applications within the flight deck environment. When combined with data from user applications, it displays

graphical images and interactive components to the flight deck crew. It also manages user-system interactions by integrating input devices for entering text (via keyboard) and for interacting with these interactive components (via mouse-like input devices). ARINC 661 emphasizes the need for independence between aircraft systems and the CDS and fully defines software interfaces between the CDS and the aircraft applications. This is a key issue in ARINC 661 as several avionics equipment manufacturers can be involved (concurrently and independently) in the development process of a cockpit while enabling easy integration through this standardisation. However, the “ARINC specification 661” standard does not specify the “look and feel” of any graphical or interaction technique information, and as such does not address human factors issues. These elements are meant to be defined by the aircraft manufacturers and/or by avionics equipment manufacturers. This means, as long as the user interface functions correctly, various aircrafts (from different families) can have different looking user interfaces, without needing to modify the User Applications.

The primary objective of the “ARINC specification 661” standard is to minimize the development costs, directly or indirectly by accomplishing the following (excerpt from [1]):

- Minimizing the cost of changing or adding new avionic systems.
- Minimizing the cost of adding new display functions to the cockpit during the life of an aircraft.
- Minimizing the cost of managing hardware obsolescence in an area of rapidly evolving technology.
- Introducing interactivity to the cockpit, thus providing a basis for airframe manufacturers to standardize the Human Machine Interface (HMI) in the cockpit.

One of the goals of the work presented in this paper is to define a software architecture clearly identifying the set of components and their inter-relations in order to reach the goals described above. Figure 1 illustrates the proposed architecture making explicit what the components are. An explanation of the detailed content of each component is out of the scope of this paper and can be found in [3].

The aim of this architecture is also to clearly identify the set of components that will be taken into account in the modelling process (via the complete description of their behaviour using ICOs). It also makes explicit the set of components that are not directly integrated in the formal description technique as the entire description of their graphical representation exploits Scalable Vector Graphic (SVG) facilities.

Such architecture presents two main advantages:

1. Every component that has an inner behaviour (server, widgets, UA, and the connection between UA and widgets, e.g. the rendering and activation functions) is fully modelled using the ICO formal description technique (see icons with a Petri net in Figure 1).

2. The rendering part is delegated to a dedicated language and tool (such as SVG or Java 2D) represented by the box closest to the user icon on the left-hand side of Figure 1.

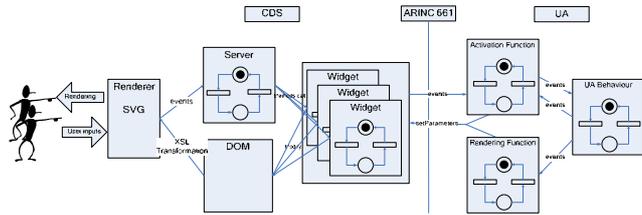


Figure 1 Detailed architecture to support ARINC 661 specification

Figure 1 is split in 3 parts. The right hand side corresponds to the user applications and the left-hand side to the widgets and the interface server. The “ARINC specification 661” mainly addresses the communication protocol between UAs and CDS (dotted line between the two parts).

In contrast to, “classical” work on interfaces for cockpits that target at extending interaction in the cockpit (as for instance in [9, 12]), the aim of ARINC 661 specification is to provide a systematic and standardised way to engineer interactive application in the cockpit.

CASE STUDY: MULTI-PURPOSE INTERACTIVE APPLICATION (MPIA)

The aim of this section is to present an application compliant with the “ARINC specification 661”. This application has simple functionality but complicated enough to highlight the main point of this paper that is to propose solutions for managing large specification using a formal notation offering a graphical representation. This User Application (UA) is made up of 3 different pages containing 12 different widgets as defined by the “ARINC specification 661”. It is important to note the usability or human factors issues are out of the scope of this paper as we focus on software engineering aspects of this application.

MPIA is a real User Application (UA) aimed at handling several flight parameters. It is made up of 3 pages (called WXR, GCAS and AIRCOND) between which a crew member is allowed to navigate using 3 buttons (as shown in Figure 2). WXR page is for managing weather radar information; GCAS is for Ground Anti Collision System parameters while AIRCOND deals with air conditioning settings.



Figure 2 Snapshots of the 3 pages of the UA MPIA

The application can be controlled (though not at the same time) by the pilot and the co-pilot via keyboard and mouse interaction. The MPIA window of any page is made up of three main parts. The information area, the menu bar and the workspace area as presented in Figure 3.

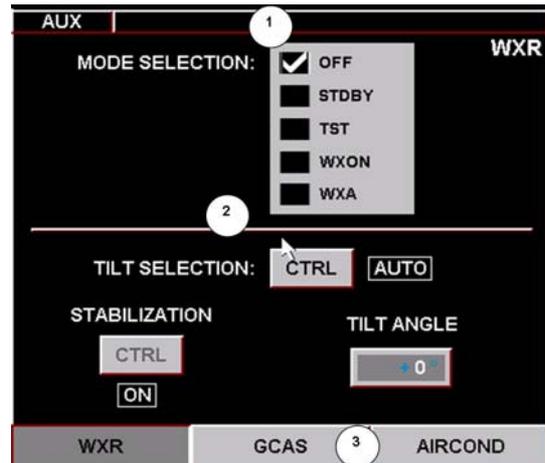


Figure 3: Multipurpose Interactive Application (MPIA)

(1) *Information area*, split in two parts, left displays current state of application, right is set by default blank but displays error messages, actions in progress or bad manipulation when necessary.

(2) *Workspace area*, display changes according to ‘interactive control panel’ selected. WXR workspace is dedicated to all modifiable parameters of the weather radar sensor. GCAS workspace is dedicated to some of the working modes of GCAS. AIRCOND workspace is dedicated to selection of temperatures inside the aircraft.

(3) *Menu bar*, includes 3 tabs for accessing 3 main ‘interactive control panels’.

ICO FORMALISM

This section recalls the main features of the ICO formalism that we used for the software modelling in the project. The models presented later correspond to the case study presented in the previous section. The Interactive Cooperative Objects (ICOs) formalism is a formal description technique dedicated to the specification of interactive systems [13]. It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance, client/server relationship) to describe the structural or static aspects of systems, and uses high-level Petri nets [10] to describe their dynamic or behavioural aspects. The extensions presented in the next section have been developed in order to tackle the issues that have been raised while working in the field of real time command and control systems (like Air Traffic Workstations, satellite ground segments [17] and cockpits [14]). We encourage the interested reader to look at [4, 20] for a complete presentation of the underlying principles of this formal description technique.

Overview of Petshop Environment

Currently, PetShop is linked to JBuilder environment for the creation of the presentation part of the ICOs (corresponding to the graphical representation of the user interface). A well-known advantage of Petri nets is their executability. This is highly beneficial to our approach, since as soon as a behavioural specification is provided in term of ObCS, this specification can be executed to provide additional insights

on the possible evolutions of the system. This way of interacting with the specification during the execution of the application as well as the numerous advantages it provides for the designers and the users has been presented in detail in [15]. The graphical nature of ICOs makes it very efficient for visualising the evolution of a system even though the behaviour of the system is highly concurrent. Similarly, with Petri nets, it is also possible to address systems with an infinite number of states. However, the graphical nature of ICOs raises management issues when handling real-size applications like the MPIA.

The next section presents some limitations that we have identified in handling models. They both lie in the structuring of models and in the graphical nature of ICOs.

Limitations of ICO & problems encountered

One of the main problems we faced and reported in [14] using the ICO notation for specifying ARINC 661 widgets is the fact that graphical presentation of widgets (called rendering) was done using Java 2D code and that that code was directly associated to each widget. The issue is related to the fact that interactive components should have a “similar” graphical representation and thus a significant part of the code dedicated to rendering appears in the code of each widget. Similarly, when a modification has to be made with respect to this graphical aspect, it has to be replicated in various places.

The number of arcs in a large model, such as that of the interface server is huge). Often, the arcs have long labels due to the large number of variables representing the parameters of different widgets that is imposed by the “ARINC specification 661”. The labels are often illegible and disrupt visualisation of the behaviour of the network. However it is not necessary that this information remains visible at all times during editing.

Also, in relation to legibility of models, a recurring problem identified during the modelling process of large systems is the need for numerous transitions to share the same input or output places. This causes regions of the model to have a high density of arcs some of which must cross the entire model to reach the destination place. This modelling problem is specific to the “ARINC specification 661” that defines, for each widget, a set of parameters whose values influence its behaviour. As (using the ICO formalism) parameters are stored as tokens in the places of the Petri net, each state changing operator (modelled as transitions) that needs to change or test the value of the token will feature an input and/or output arc from the place.

The fourth problem encountered relating to the visualisation of large models was their size, making it difficult, and sometimes even impossible to view them entirely on the screen. This is why one seldom works on such models with a level of so significant zoom, for rather privileging a level making readable the information. We thus had to extend PetShop to simultaneously provide a focus and context view on the models.

EXTENDING ICO AND PESHOP TO COPE WITH SCALABILITY ISSUES

This section details how the ICO formalism has been extended in order to take into account the specificities of the “ARINC specification 661” components (UAs, server and widgets). Even though we mainly detail these modifications with respect to the ICO formalism any other graphical notation like Statecharts [11] or automaton [21] would require similar extensions to deal with this kind of application domain. The contribution of this section is not the solution per se but the issues raised and how the solutions proposed and implemented have been able to solve the problems and thus to allow ICOs to scale up to handle large interactive software systems.

Restructuring models for graphical appearance

The goal of model restructuring is twofold; firstly it simplifies the management of graphical aspects, reduces the size of the models and reduces development time. Secondly, it provides an elegant solution to a recurring problem related to the selection of widgets, called “picking”. Picking is a task that has to be solved by the user interface server in order to determine which widget has been the target of a user action using the input device.

Initially we decided to handle picking and graphical rendering in a similar way as we did in other application domains (like air traffic control and satellite ground segment applications). Picking was explicitly modelled using ICO in the server model while rendering was handled by including Java/2D code in the rendering function of the ICO models (as presented in [14]). However, these two ways of handling the problem have raised more problems when dealing with the large scale application. We thus defined a structuring method of these specific aspects by using SVG (Scalable Vector Graphics) for rendering and by externalizing picking away from the server behaviour. We do not present here in full detail the new solution. However, the following tables show the benefit that resulted with the restructuring presented above.

Type of widget	ARINC 661 Widget	Size of graphics code (Java/2D)	Size of graphics code (SVG)
Container	RadioBox	142 lines	15 lines
	TabbedPanelGroup	168 lines	16 lines
	TabbedPanel	151 lines	25 lines
	Panel	152 lines	16 lines
Graphical	GPTriangle	221 lines	15 lines
	GPRectangle	178 lines	15 lines
Interactive	PicturePushButton	333 lines	55 lines
	PictureToggleButton	393 lines	55 lines
	CheckButton1	399 lines	60 lines
	Label	216 lines	16 lines
	LabelComplex	379 lines	25 lines
	EditBoxNumeric	490 lines	50 lines

Table 1 Measure of volume of each widget in terms of number of lines of code for the graphical part. (left Java2D/right SVG)

A detailed presentation of the design process exploiting this restructuring of model is available in [16]. Table 1 shows a measurement of the volume of each widget in terms of a number of lines of codes for the graphical part, and Table 2, in terms of a number of places and transitions.

Widget	Model size		Model size	
	Separated picking	Centralised picking	Places	Transitions
RadioBox	49	29	28	21
TabbedPanelGroup	62	22	44	16
TabbedPanel	72	49	22	7
Panel	65	46	16	5

Table 2 Measure of volume of each widget in terms of model size (left Java2D/right SVG)

Label hiding

A simple solution to the arc label problem is to give the designer the option to mask arc labels. This modification is purely graphical and does not change any modelling aspect. Once the arc labels are masked, it is possible to edit the arcs in the usual way (double clicking on the desired arc). Figure 4 and Figure 5 illustrate the difference between displayed and masked arcs. The point here is not to read the models but to see that the flow of information (arcs between places and transitions) is much easier to perceive with hidden arc labels.

Virtual places

In the server ICO model, one place contains references of all the interactive widgets of the application executed by the server. This implies that all transitions creating an interactive widget or using an interactive widget has an arc linked to this place (in the server model for the MPIA application resulting in more than 50 arcs).

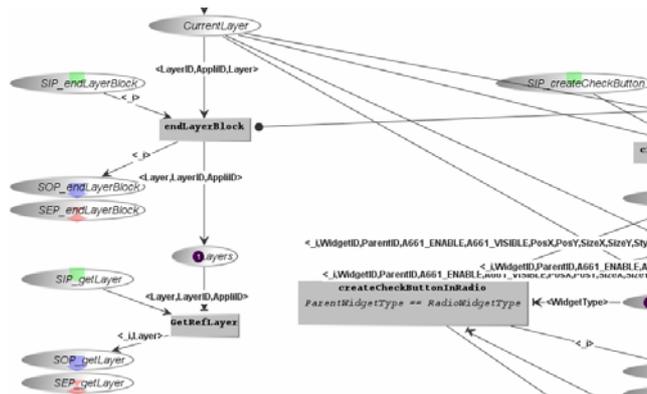


Figure 4 Partial view of the MPIA Server with displayed arc labels

The ICO formalism is defined following an object oriented structuring of models. This means that models are usually quite limited in size as they only address the behaviour of a single object. While dealing with the “ARINC specification 661”, the server cannot be split in several sub objects as (as stated above) references to the widgets are required for most of the actions. In order to deal with the complexity of the resulting server model we have defined and implemented the

notion of virtual places in a Petri net model. These virtual places make it possible to define regions in a model. Each region is responsible for a given part of the behaviour. Virtual places allow exploiting shared parameters between regions improving the legibility of models.

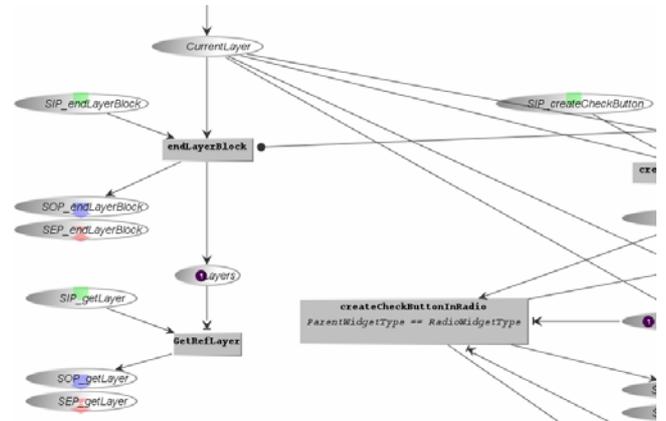


Figure 5 Partial view of the MPIA Server with arc labels masked

A virtual place is a partial copy of a normal place. It adopts the display properties (such as markings) but not the arc connections. The arcs can be connected to normal places or to virtual places.

The display is therefore modified allowing easier reorganisation of models. On a semantic level, the group made up of the “source” and its virtual places, behaves like a single place. For instance, if one token is removed by a transition for one of these places, then it will be removed from all of them. The left hand side of Figure 6 shows three places P1 (the “source” place with a darker border and the two virtual places) each with an arc. The right hand side of Figure 6 shows the result of the reconstitution of the place

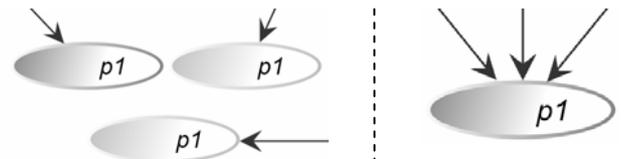


Figure 6 Real view (left) versus Semantic view (right)

This functional structuring of models and the support of virtual places significantly reduces the time and effort for construction and modification of models. It is however important to note that this kind of construct has to be used carefully as, breaking down the graphical dynamics of the models can also result in making it more difficult to understand their behaviour. To solve this problem we have defined usage rules limiting their use to places storing parameters and which are not part of an information flow.

Mini view

In order to simplify the navigation in large models, we have added a “mini map” feature to Petshop. The bird’s eye view is a representation of the complete model but on a much smaller scale. A screenshot of Petshop including the bird’s eye view is not presented due to space constraints. The

bird's eye view contains a small box which highlights what is currently visible in the main viewing area. The user can change which part of the model is visible in the main viewing area by moving a small box on the bird eyes view. On the contrary, if the user zooms or move within the main viewing area, the box on the bird's eye view will move accordingly.

CONCLUSION AND PERSPECTIVES

In this paper we have presented problems incurred when trying to model large, real life embedded safety-critical interactive applications systems using the ICO formalism and the Petshop tool. We approached the structural modelling problems in a systematic way to deal with scalability by making modifications on two levels; modification of the tool to facilitate the management of large models and modification to the structuring mechanisms of the modelling approach.

The specification and modelling parts of the study have shown that it is possible to formally model the entire behaviour of a real-life safety-critical interactive application. The interactive cooperative objects (ICO) formal description technique has been adapted to meet the requirements of user applications compliant with ARINC 661. This adaptation has been applied to the MPIA application.

The same description technique was applied for the modelling of ARINC 661 components and the server. One of the difficulties for the server side was the multiple instantiation of the various components embedded in the MPIA interactive application but also the modelling of the interaction techniques and the management of the various input sub-systems like mouse and keyboard.

The specification technique was additionally successfully applied to a subset of the 57 components described in the "ARINC specification 661". This subset covers all components necessary for the MPIA but also other ones featuring more complex behaviour such as Popup menus or ComboBoxes.

Modelling interactive applications requires the creation of numerous models (even relatively simple types as the MPIA requires 37 models and 91 instances). Without an adequate tool that allows the entry of data as well as the simulation of models and without an adequate modelling process, this task would not be possible due to industrial resource constraints (human, time and financial).

After working on the expressive power of the ICO notation and the definition of a CASE tool supporting the various activities related to the construction and modification of ICO models, we have extended and improved the editing and simulation features of Petshop to increase the applicability power of the notation. Indeed, Petshop and ICOs make it possible to manage large scale models by means of various interaction techniques including a bird's eye view, management of "virtual places" and management of arc labels.

In addition to the above mentioned results, the study also provided interesting prospects. In an interactive cockpit

application like MPIA, we noticed that both the pilot and co-pilot are interacting with the same application. In the current deployed version of MPIA multi-user interaction is not allowed as only the first cursor entering a window is active (it becomes inactive after a period of inactivity). However, taking into account multi-user interaction techniques which allow collaborative interaction between the pilot and co-pilot might become a requirement for future cockpit systems. We have already shown the capability of ICOs and PetShop for dealing with multimodal interactions [16] but we also believe that addressing that aspect within real-size applications might raise additional challenges.

ACKNOWLEDGEMENTS

This work was supported by the EU funded ADVISES Research Training Network, GR/N 006R02527. <http://www.cs.york.ac.uk/hci/ADVISES/>. The work is also partly funded by DPAC (Direction des Programmes de l'Aviation Civile) under contract #00.70.624.00.470.75.96. Special thanks are due to our colleagues at Thales and especially P. Cazaux. PetShop development has also been supported by DGA (French Army Research Dept.) under contract INTUITION #00.70.624.00.470.75.96.

REFERENCES

1. ARINC 661, Prepared by Airlines Electronic Engineering Committee. Cockpit Display System Interfaces to User Systems. Arinc Specification 661. 2002.
2. ARINC 661-2, Prepared by Airlines Electronic Engineering Committee. Cockpit Display System Interfaces to User Systems. Arinc Specification 661-2; 2005.
3. Barboni E, Conversy S., Navarre D. & Palanque P. Model-Based Engineering of Widgets, User Applications and Servers Compliant with ARINC 661 Specification. Proceedings of the 13th conference on Design Specification and Verification of Interactive Systems (DSVIS 2006), Dublin, Ireland, July 2006, Lecture Notes in Computer Science, Springer Verlag.
4. Bastide, R and Palanque, P. A Petri-Net Based Environment for the Design of Event-Driven Interfaces 16th International Conference on Applications and Theory of Petri Nets, ICATPN'95: Lecture Notes in Computer Science 935; 1995: 68-83.
5. Bastide, R and Palanque, P. A Visual and Formal Glue between Application and interaction. International Journal of Visual Language and Computing. 1999; 10(5):481-507.
6. Booch G., J. Rumbaugh, I. Jacobson: The Unified Modeling Language Reference Manual, Addison-Wesley, 1999.
7. Degani A., & Heymann, M. (2003). Analysis and Verification of Human-Automation Interfaces. Human Centered Computing: Cognitive, Social and Ergonomic Aspects, Vol.3, pp. 185-189. Mahwah, NJ: Erlbaum. (Proceedings of the 10th International Conference on Human - Computer Interaction, Crete, June 22-27, 2003)
8. Degani, A., Heymann, M., (2000). Pilot-autopilot interaction: A formal perspective. In Abbott, K., Speyer, J.J., Boy, G., eds.: Proceedings of the International Conference on Human-Computer Interaction in Aeronautics: HCI-Aero 2000, Toulouse, France, pp. 157-168.

9. Faerber R. Vogl T. & Hartley D. Advanced Graphical User Interface for Next Generation Flight Management Systems. In proceedings of HCI Aero 2000, pp. 107-112.
10. Genrich, H. J. Predicate/Transitions Nets, High-Levels Petri-Nets: Theory and Application. K. Jensen & G. Rozenberg (eds.). High-Level Petri: Theory and Applications: Springer Verlag: 3-43., 1991.
11. Harel, D. Statecharts: A visual formalism for complex systems. Science of Computer Programming; 8, 231-274 (1987).
12. Marrenbach J. & Kraiss K-F. Advanced Flight Management System: A New Design and Evaluation Results. In proceedings of HCI Aero 2000, pp. 101-106.
13. Navarre, D., Palanque, P., Bastide, R. A Tool-Supported Design Framework for Safety Critical Interactive Systems in Interacting with computers, Elsevier, Vol. 15/3, pp 309-328, 2003.
14. Navarre, D; Palanque, P, and Bastide, R. A Formal Description Technique for the Behavioral Description of Interactive Applications Compliant with ARINC 661 Specifications. International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero'04); Toulouse, France. 2004.
15. Navarre, D; Palanque, P. & Bastide, R. Reconciling Safety and Usability Concerns through Formal Specification-based Development Process HCI-Aero'02 MIT, USA, 23-25 October, 2002.
16. Navarre, D., Palanque, P., Dragicevic P. & Bastide, R. An Approach Integrating two Complementary Model-based Environments for the Construction of Multimodal Interactive Applications. Interacting with Computers, vol 17, n°2, (to appear), 2006.
17. M. Ould, R. Bastide, D. Navarre, P. Palanque, F. Rubio, A. Schyn. Multimodal and 3D Graphic Man-Machine Interfaces to Improve Operations. Eighth International Conference on Space Operations, Montréal, Canada, May 17 - 21, 2004. <http://www.spaceops2004.org/papers>
18. Sherry L., Polson P., Feary M. & Palmer E. When Does the MCDU Interface Work Well? Lessons Learned for the Design of New Flightdeck User-Interface. In proceedings of HCI Aero 2002, AAAI Press, pp. 180-186.
19. Shiffman, S., Degani, A., Heymann, M (2004) UIVerify - a Web-Based Tool for Verification and Automatic Generation of User Interfaces. International Conference on Human-Computer Interaction in Aeronautics HCI-Aero 2004 29 September to 1 October 2004 Toulouse, France.
20. Sy, O; Bastide, R; Palanque, P; Duc-Hoa, KL, and Navarre, D. PetShop: a CASE Tool for the Petri Net Based Specification and Prototyping of CORBA Systems. 20th International Conference on Applications and Theory of Petri Nets. ATPN '99; Williamsburg, VA, USA. LNCS Springer Verlag; 1999.
21. Wood W.A., "Transition network grammars for natural language analysis", Communication of the ACM vol. 13 n°10. October 1970 p. 591-606.

Software Hazards & Barriers for Informing the Design of Safety-Critical Interactive Systems

S. Basnyat

LIHS-IRIT, University Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France

P. Palanque

LIHS-IRIT, University Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse Cedex 9, France

ABSTRACT: This paper presents a systematic approach to formally model and analyse socio-technical barriers for safety critical interactive software applications by first identifying interaction hazards. Our approach is grounded on current knowledge in the field of hazard identification, barrier analysis and modelling and its application is shown in a safety critical context.

1 INTRODUCTION

In the aviation industry, cockpits are becoming increasingly interactive with the introduction of the keyboard cursor control unit (KCCU) combined with graphical display units (DU) to be embedded in the Airbus A380 for commercial flights in 2006 and later in the Boeing 787 aircraft.

There exists extensive literature on types of human computer interaction failures that can occur while a user is operating an interactive system. Examples include mode confusions which are a kind of automation surprise (Rushby et al. 1999). Evidently, the consequence of such “errors” in a safety-critical context can potentially be devastating. The roots of such “errors” are often the result of poor design. Typically, there is a mismatch between the designer’s conception of the system and the user’s interpretation of the system. From an organizational perspective, such “errors” could also be a consequence of poor training, management etc.

Ideally, the possible existence of these “errors” should be considered throughout the design process, as early as during requirements gathering to mitigate them and minimize development costs. Techniques for analyzing software safety such as Preliminary Hazard Analysis (PHA) including Fault Tree Analysis (FTA) and System and Subsystem Hazard Analysis (SHA and SSHA) including deviation analysis such as Failure Mode Effects and Criticality Analysis (FMECA), mode confusions analysis and state machine hazard analysis are targeted for such early analysis. (Leveson et al. 1997).

The identification of potentially dangerous software aspects can be used to inform changes to the design, and training procedures etc. With respect to the design, such modifications could be considered as software barriers.

While barriers have been used quite intensively in the field of hardware and socio-technical systems (Hollnagel 1999) their use in the field of

software systems remains very limited as for instance in (Leveson 1991). The intrinsic nature of interactive software systems that involve users in their daily operation require the same level of reliability but, surprisingly, a barrier approach has not been applied to this area so far. This might be related to the fact that interactive applications have been kept away from command and control safety critical areas or to the fact that redesigning the system has been considered as a more adequate alternative than integrating barriers to the extant system.

A barrier is an obstacle, an obstruction, or a hindrance that may either (i) prevent an action from being carried out or an event from taking place, or (ii) prevent or lessen the impact of the consequences, limiting the reach of the consequences or weakening them in some way (Hollnagel 1999). According to Leveson (Leveson 1991), a distinction can be made between three types of barriers called lockout, lockin, and interlock, respectively. A lockout device “prevents a hazardous, event from occurring, while a lockin device maintains safe conditions. An interlock is used to ensure that a sequence of operations occurs in the correct order.”

This paper presents a complementary approach for the systematic identification of software-related user interaction hazards and barriers and modelling of barriers. The basis of sound design for a safety-related system is the identification, through systematic analysis, of the hazards which the system might encounter in operation as claimed in (Falla 1997). The proposed approach provides systematic ways to deal with the increase of reliability of safety critical interactive applications.

The design of a usable, reliable and error-tolerant interactive safety-critical system is a goal that is hard to achieve but can be more closely attainable by taking into account information from previous usages of the system. One such usually available and particularly pertinent source is the outcome of an incident or accident investigation.

Designs of any nature can be improved by taking into account previous experiences, both positive and negative. However, it is not always the case that an accident report will be available, especially when new technology is involved. With a more pro-active perspective, we aim to analyse a safety critical interactive application, similar to those that will be embedded in the new interactive cockpits (like the Airbus 380 cockpit or Boeing 787) that are compliant with the ARINC 661 specification, for which we have no accident data. We aim to identify flaws in the application and suggest software barriers that could minimise potential erroneous user interaction with the system.

With the objective of increasing safety, in safety-critical interactive systems, previous research in the field aimed at trying to eliminate the error completely by identifying its source. It has now been widely accepted however that human errors are inevitable due to the unpredictable behaviour of humans and we must instead try to manage errors. The perspective of blame has also changed from isolating an individual operator to having a wider outlook on the organisation as a whole. However, the broader the perspective, the more information has to be gathered and thus making it more complex not only to organise it but also to reason about it.

In order to tackle these issues, modelling processes and techniques have been defined and applied widely in the field of safety critical systems. Model-based development (MBD) is a developing trend in the domain of software engineering (MDA Guide version 1.0.1 2003) advocating the specification and design of software systems from declarative models (Puerta 1998). It relies on the use of explicit models and provides the ability to represent and simulate diverse abstract views that together make up a 'system', without the need to fulfill its implementation. It is widely accepted within the community that models are needed for the design of safety critical interactive systems; this is to be able to understand issues such as safety and resilience and to think about how safety can be ensured, maintained, and improved (Hollnagel and Woods 2006).

Previously, we focused on task models and error analysis of sub-tasks to identify potential problems (See (Palanque and Basnyat 2004) for cash machine PIN entry example, & (Basnyat et al. 2005b) for a hardware-related valve opening example). We argue that although it is important for human "error" analysis and consideration for human factors to be included throughout and in early stages of the development process, further problems can occur during the interaction with the system.

In addition to the skill-based Human Error Reference Table (HERT) defined and exemplified

in our previous work, we apply a typical heuristic evaluation to the UI and a Hazard and Operability Studies (HAZOP) analysis. The identified interaction hazards are used to propose socio-technical barriers to mitigate these hazards. The proposed barriers, which may take the form of improved training, software implementation modifications, or UI design improvements for example, are also categorised according to Leveson's (Leveson 1991) and Hollnagel's (Hollnagel 1999) barrier classifications. By doing so, the barriers can be filtered and represented in their relevant design model. For example, barriers relating to the behaviour of the software can be represented in the system model whereas barriers relating to improved training could modify the operator's task and thus be modelled in the task model. Barriers relating to warning signs, personal protective equipment or fire extinguishers for example should be represented in a dedicated model.

In this paper, we focus on barriers related to the behaviour of the system. The relevant software barriers are extracted from the identified list and are modelled using the Interactive Cooperative Objects (ICO) (Navarre et al. 2003), a formalism dedicated to the description of interactive systems based on a dialect of Petri nets.

The approach has been applied on an interactive cockpit application called Multi Purpose Interaction Application (MPIA) whose formal description has been presented in (Navarre et al. 2004). The following section briefly describes the case study. Section 3 details the interaction hazard analyses and results followed by section 4 which presents the proposed barriers. Finally, we present the modelling of the barriers and their integration with the previously modelled system model of the interactive cockpit application in section **Erreur ! Source du renvoi introuvable.**

2 CASE STUDY: MULTIPURPOSE INTERACTIVE APPLICATION (MPIA)

The aim of this section is to present the case study, the MPIA User Application (UA) compliant with the ARINC 661 specification. The Airlines Electronic Engineering Committee (AEEC) (an international body of airline representatives leading the development of avionics architectures) formed the ARINC 661 Working Group to define the software interfaces to the Cockpit Display System (CDS) used in all types of aircraft installations. The standard is called ARINC 661 - Cockpit Display System Interfaces to User Systems (ARINC 661-2 2004).

MPIA is a User Application (UA) that aims at handling several flight parameters. It is made up of

3 pages (called WXR, GCAS and AIRCOND) containing 12 different widgets as defined by the ARINC 661 specification. The crew member is allowed to navigate between the 3 pages using 3 buttons (as shown in Figure 1). The WXR page is responsible for managing weather radar information; GCAS is responsible for Ground Anti Collision System parameters while AIRCOND deals with settings of the air conditioning.

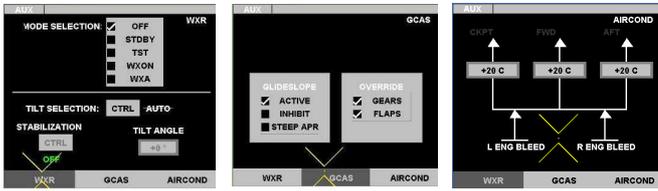


Figure 1 Snapshots of the 3 pages of the MPIA UA

The application can be controlled (though not at the same time) by the pilot and the co-pilot via keyboard and mouse interaction. Each MPIA window contains three main parts: The information area (on the top), the menu bar (on the bottom) and the workspace (central part).

(1) *Information area*, split in two parts, left displays current state of application, right is set by default blank but displays error messages, actions in progress or bad manipulation when necessary.

(2) *Workspace area*, display changes according to ‘interactive control panel’ selected. WXR workspace is dedicated to all modifiable parameters of the weather radar sensor. GCAS workspace is dedicated to some of the working modes of GCAS. AIRCOND workspace is dedicated to selection of temperatures inside the aircraft.

(3) *Menu bar*, includes 3 tabs for accessing 3 main ‘interactive control panels’.

3 SOFTWARE SAFETY & INTERACTION HAZARD IDENTIFICATION

This section is dedicated to the description of the interaction hazard identification analyses including related work in the field of software safety analysis. In (Falla 1997), it is argued that current software safety standards provide little advice regarding threats which ought to be considered when undertaking software hazard analysis such as environmental and operating conditions, logic control, real time executive, system function calls, system resources, timing and software design notations. Although all these types of hazards are important, in this paper we focus our attention on hazards relating directly to UI interpretation problems and ways of improving the UI to assist the operator. It was found in (Falla 1997), that “optimum [software safety analysis] results were obtained when techniques including Fault Tree Analysis (FTA), Failure modes and effects analysis

(FMEA) and Hazard and Operability Studies (HAZOP) were used in a coherent fashion as part of an integrated safety assessment method” p.6 chapter 3. Although the authors apply HAZOP to an existing software design, the approach is based on data flow diagrams. Our approach differs since we apply the HAZOP analysis more specifically to the user interface.

In closer relation to our approach, Leveson (Leveson et al. 1997) presents an approach for analysing the safety of the Center-TRACON Automation System (CTAS) portion of an Air Traffic Control (ATC) system. The approach aims to analyse the effect the system changes have on human “errors” in order to modify the system and improve user training to reduce their impact. The approach looks at the automation as a way of evaluating its potential to contribute to human “error”. The focus of this work is on automation and on mode confusion. Six “categories of potential design flaw” are identified including interface interpretation error, inconsistent behaviour, indirect mode changes, operator authority limits, unintended side effects and lack of appropriate feedback. Our approach builds and extends this kind of work as it does not focus on only mode confusion but on human-computer interaction failures because of cognitive “errors” and poor design.

3.1 Skill-based HERT analysis

Human error plays a major role in the occurrence of accidents in safety-critical systems such as in aviation, railways systems, or nuclear power plants (Reason 1990).

Although the term “human error” appears very controversial, theories of human “errors” such as Rasmussen’s (Rasmussen 1983) SRK, Hollnagel’s (Hollnagel 1991) Phenotypes and Genotypes and Norman’s (Norman 1990) classification of slips can be considered widely acceptable.

We have produced Human Error Reference Tables (HERTs) that are in early phases of development and the current version is available on the following web site <http://liihs.irit.fr/basnyat>. The three tables are based on the above human “error” theories. The benefit of producing such reference tables is the support it offers for the exact identification of precise types of ‘error’ while analysing human behaviour.

In (Palanque and Basnyat 2004) and (Basnyat et al. 2005b) we applied HERTs to subtasks of a user task model. However, this kind of task-based analysis of deviation requires detailed information concerning pilot or co-pilot tasks involving the MPIA. In this paper we propose a complementary approach using HERTs that is based on generic

Table 1. Extract of Skill based Human Error Analysis Results on Weather Radar (WXR) page

Under-motivation	Try to change tilt angle without first changing tilt selection mode to manual	Tilt angle is unmodifiable
Description error	Click on stabilization button instead of tilt selection button	If tilt selection was in manual, then stabilization mode is available and becomes on or off depending on its current state If tilt selection was in auto, then stabilization mode is unmodifiable
	Change tilt angle in a temperature character box	Temperature will be set to figure entered or a figure nearest that entered if outside the 18-28 degree range and tilt angle will remain the same Will cause delay in interaction
Input or Misperception errors	Not realise that stabilization mode can only be changed when tilt selection is in manual	
Data driven error	Enter a tilt angle figure of a different figure that is currently being looked at or discussed	Tilt angle will be set to the degree entered or a degree nearest to that entered if outside of the +15 -15 degree range.
Order Errors	Changing tilt angle before changing tilt selection mode	Tilt angle is unmodifiable
...

“error” types and does not require this kind of

We have applied the HERTs directly to the Weather Radar (WXR) part of the application without using any information from pilot’s tasks. Using HERTs we have been able, for each type of human error, to deduce possible erroneous interactions with the system and what the impact of the identified ‘error’ will be on the task in hand. The aim of these reference tables is not to guarantee a comprehensive and exhaustive identification of every possible eventuality but to provide investigators with systematic ways of exploring likely and reoccurring user deviations. For example, Table 1 documents an extract of the outcome of this process focussing on the skill-based errors. Due to space constraints we have not presented the other two i.e. rule-based and knowledge-based. The key benefit of this human ‘error’ analysis is that its style not only helps to identify potential interaction problems that led to a particular incident (if analysing after an incident), it can also be used to identify other alternate problems that might jeopardise the future safety of an application but which did not arise in this specific accident.

3.2 Heuristic analysis

Heuristic Evaluation (HE) (Nielsen and Molich 1990) is an established Usability Inspection Method (UIM) in the field of HCI (Human-Computer Interaction). The analyst, who must have knowledge in HCI, follows a set of guidelines to analyse the user interface. The technique is inexpensive, easy to learn and can help predict usability problems early in the design phase if prototypes are available. Examples of HEs include (Nielsen and Molich 1990), (Pierotti 1995) and (Gerhardt-Powals 1996). We have chosen to use Nielsen’s 10 usability heuristics due to the nature of MPIA application whose interaction and visualization techniques correspond directly to the type of desktop application targeted by this kind of heuristics. It has been argued that such a UIM is not the most

effective for identifying potential usability problems (Cockton et al. 2003). We understand that for larger, more complex systems involving complex activities from users, it would be more appropriate to base the evaluation on a user goal and involve analysts to predict usability as we have done while identifying potential scenarios of interaction with a system (Basnyat et al. 2005a).

Due to space constraints we cannot present all of the results of the heuristic evaluation. An extract is presented in Table 2. The results have been used to identify potential software barriers. By this we do not mean barriers to accessibility as in (Killam and Holland 2003), instead we mean potential software barriers that could help prevent erroneous

Nielsen’s 10 Heuristics	Result
Visibility of system status	Mode selection status clear, tick box indicator & selection highlight Tilt selection, Stabilization and Tilt angle status less clear
Match between system and the real world	Abbreviations could be misleading to non-experienced pilot Layout of cockpit, forward cabin and afterward cabin temperature setting displays do not match real world layout of aircraft (AIRCOND)
User control and freedom	No undo/redo however user can easily undo/redo their actions
Consistency and standards	The checkbox widget for mode selection behaves as a radio button widget. With respect to the other 2 tabs, GCAS & AIRCOND, the checkbox widget does not behave in the same way as the checkbox widget of GCAS Override which allows 2 boxes to be selected at the same time.
Error prevention	No confirmation options
Recognition rather than recall	No recall necessary except for abbreviations
Help users recognize, diagnose, and recover from errors	None available
Help and documentation	None available
...	...

user interactions.

Table 2. Extract of Heuristic Evaluation Results (mainly for WXR page unless otherwise stated)

3.3 HAZOP analysis

Although a full explanation of the HAZOP technique is outside the scope of this paper, the interested reader can see (Kletz 1999). Briefly however, HAZOP is a systematic technique that attempts to consider events or process in a system exhaustively. Within a particular system domain (or scenario), items (or events) are identified and a list of guidewords is applied to the items. The guidewords prompt consideration of deviations to item behaviour (guideword examples include less, more, none, more than) to elicit the potential for arriving at possible hazardous states (Smith and Harrison 2002). These guidewords provide the structure of the analysis and can help to ensure complete coverage of the possible failure modes (Pumfrey 1999). Although a HAZOP analysis should generally be applied to a Piping and Instrumentation Diagram, we apply the analysis to the WXR UI. An extract of the results can be seen in Table 3.

Table 3 Extract of HAZOP analysis results on WXR tab

Guideword	Deviation	MPIA WXR Interpretation
MORE THAN or AS WELL AS	All intentions achieved, but with additional effects (qualitative increase)	Change tilt angle, also change stabilization mode to off
PART OF	Only some of the intention is achieved (qualitative decrease)	Change tilt selection to manual, but do not change tilt angle
OTHER THAN	A result other than the intention is achieved	Mistype new tilt angle but do not realize, new state is undesired
REVERSE	The exact opposite of the intention is achieved	Tilt angle is not entered/changed
Early, Late, Before & After		Not analysed, could be applicable to procedures of the pilots
...

4 BARRIERS

In this section we discuss the relation between the identified interaction hazards and potential barriers that could be imposed to minimise the threats. We briefly discuss barrier analysis and then detail the categorisation of proposed barriers according to Hollnagel and Levesons' categorisations.

Barrier analysis is often associated with accident analysis in aiming to determine which barriers failed, why they failed and how to rectify the problems. "Barrier analysis starts from the assumption that a hazard comes into contact with a target because barriers or controls were unused or inadequate" (Johnson 2003) p.355).

A barrier is an obstacle, an obstruction, or a hindrance that may either (i) prevent an action from being carried out or an event from taking place, or (ii) prevent or lessen the impact of the consequences, limiting the reach of the consequences or weakening them in some way (Hollnagel 1999). There are several distinctions of barrier types. According to Johnson (Johnson

2003), there are three different forms of barriers, people, process and technology. Hollnagel (Hollnagel 1999) distinguishes between four types of barriers, Material barriers, barriers that physically prevent an action of limit the negative consequence of an event; functional barriers, barrier that logically or temporally link actions and events; symbolic barriers, barriers that require interpretation and immaterial barriers, barriers that are not physically in the work situation. It can be seen that the above classifications relate more closely to large hardware related domains, such as chemical engineering plants. Little research has been carried out on barrier analysis and classification for software applications except (Leveson 1991). According to Leveson, a distinction can be made between three types of barriers called lockout, lockin, and interlock, respectively. A lockout device "prevents a hazardous, event from occurring, while a lockin device maintains safe conditions. An interlock is used to ensure that a sequence of operations occurs in the correct order."

4.1 Barrier Identification

This section is dedicated to the discussion of human-computer interaction/software barrier identification. Previously (Basnyat et al. 2005b), (Schupp et al. 2006) we used incident and accident reports to identify existing (successful and failed) socio-technical barriers. The case study in this paper differs from our previous work in that it is a software application and there exists no known accident report with reference to this particular application. This section describes the barriers that were identified following the three analyses.

The barriers identified following the heuristic analysis relate mainly to the interface design, layout and representation. The first barrier (B1) (see Table 2 visibility of system status) is related to graphical feedback provided by the system and should improve user understanding of the current state of the system. For example, the current status of tilt selection, stabilization, tilt angle and Glidescope selection are not clear. A second barrier (B2) (see Table 2 match between system and real world) could be an improvement in the design of the AIRCOND page such that the layout of the modifiable temperatures resembles the layout of the aircraft to reduce possible misinterpretation. In the GCAS page there is a restriction on selecting more than one 'GLIDESCOPe' option at any given time. Options include ACTIE, INHIBIT or STEEP APR. However, in relation to consistency and standards, symbolic coding should be imposed (B3) (see Table 2 consistency and standards) such that radio style button interaction is represented as radio buttons and not as checkboxes. Error prevention is

non-existent in the application. Symbolic warning (visual or auditory for example) is required (B4) (See Table 2 Error prevention) for each mode change, tilt angle changes and temperature changes that could conflict with other modes and settings. However, it is not necessarily useful to have a warning for every modification. Providing reference (B5) (See Table 2 Recognition rather than recall) to abbreviations could help reduce possible misinterpretation of labels used. Furthermore, a combination of training (non-technical barrier) and software feedback could help users recognize, diagnose, and recover from errors.

To minimise the likelihood of the issues relating to the categories, detached intentions, under motivation, description error, input or misperception, omission, reversal, insertion and order errors, a barrier relating to the three tabs of the interface should be imposed (please note, not all of these results are available in Table 1). In relation to the description error issue, the barrier (B6) (See Table 1 description error) should enforce clear distinctions between button behaviour. For example, the Tilt selection mode and stabilization mode buttons are both labelled CTRL however the behaviour of CTRL for Tilt selection is auto/manual and the behaviour of CTRL for stabilization mode is on/off.

Table 4 MPIA software barrier classification

No	Barrier	Analysis Technique Identifier	Leveson's software barrier classification	Hollnagel's system barrier classification	
B1	Improve user understanding of current system state	Heuristic	N/A	Symbolic	Indicating
B2	Improve design of AIRCOND workspace such that the layout of the modifiable temperatures resembles the layout of the aircraft	Heuristic	N/A	Symbolic	Indicating
B3	Use radio style button interaction for GLIDESCOPE option in GCAS workspace	Heuristic	Interlock	Symbolic Functional	Indicating Soft Preventing
B4	Symbolic warning is required for all mode changes, tilt angle changes and temperature changes.	Heuristic HAZOP	Lockout	Symbolic Functional	Indicating, Countering, Preventing, Hindering
B5	Provide references and ensure correct training regarding abbreviations.	Heuristic	N/A	Symbolic Immaterial	Indicating Prescribing
B6	Enforce clear distinctions between button behaviour	HERT	N/A	Symbolic	Indicating, Countering
B7	Provide a symbolic warning in relation to the figures entered and acceptable figures.	HERT + HAZOP	Lockout	Symbolic	Indicating Countering
B8	If tilt angle is not entered, system should not allow other actions to be available until task in hand is complete i.e. tilt angle entered	HAZOP	Lockin	Functional	Soft Preventing, Hindering
B9	When sequences of actions should be performed in a certain order, the UI should guide the user	HAZOP	Interlock	Functional	Soft Preventing

The data driven error example could be minimised with a barrier (B7) (See Table 1 data driven error) providing a symbolic warning in relation to the difference of the figure entered and the acceptable figure. There is no such barrier implemented in the MPIA. The tilt angle range (lower right hand corner of the left-hand side of Figure 1) must be between -15° and $+15^{\circ}$. If a tilt angle outside of the range is entered, no warning is provided, the angle is set to the nearest entered, i.e. if -19 is entered, the tilt angle will be set to -15° , if $+40$ is entered, the tilt angle will be set to $+15^{\circ}$. The same rule applies for the temperature setting however the range is between 18° and 28° degrees. The preventative barrier therefore would be to

inform the pilot or co-pilot that their command has not been accepted. Also it was noted that if a character is entered into the tilt angle text box as opposed to a number, the previously set angle disappears and becomes simply $+ 0^{\circ}$. We do not know the impact this would have on the operation of the application within a cockpit.

The HAZOP analysis highlighted two further barriers in addition to exemplifying barriers 4 and 7 which had already been identified. (B8) (see Table 3 part of) calls for a restriction on available interaction until the current task in hand is complete, in this case entry of a new tilt angle for the WXR. (B9) (see Table 3 more than/as well as) aids the user to follow a correct sequence of actions by modifying the UI. This would also support potential cognitive "errors" such as a post-completion error.

4.2 Barrier Classification

In total, 10 barriers (after generalising the findings) were identified using the three complementary techniques. They can be considered a combination of protective and preventative barriers. These barriers are classified (see Table 4) according to Hollnagel's classification of system barriers and Leveson's classification of software barriers. This is

particularly useful because it is possible to then allocate each type of barrier to a relevant design model for accurate representation. We are interested in functional barriers since this is where we can make most contribution to the field. The functional barriers can be modelled using the ICO notation and integrated with an existing system model of the MPIA using the same notation to induce a safer interaction system. The ICO CASE tool, Petshop supports simulation of models, thus once the barrier has been integrated, it is possible to view the impact it would have on the system and human interactions with the system.

5 BARRIER MODELLING

6 CONCLUSIONS AND FURTHER WORK

We have presented a systematic approach for identifying user interaction hazards and their related mitigating software barriers based on three complementary analysis techniques: heuristic, human error and HAZOP analysis. We have used the ICO formal description technique to model the extracted functional barriers and integrate them with an existing model of a complete real-life embedded interactive cockpit application. The modelling of such barriers allows us to simulate potential user interactions with the system to ensure that the barrier eliminates the previously identified hazards.

The approach also aids in identifying further barriers such as symbolic and immaterial barriers that should be represented and taken into account within the development process, however not within the system model.

ACKNOWLEDGEMENTS

This work is funded by DPAC (Direction des Programmes de l'Aviation Civile) étude "validation cockpit interactif", EU via the ADVISES Research Training Network RTN2-2001-00053 and Network of Excellence ResIST (www.resist-noe.org).

REFERENCES

1. ARINC 661-2. Cockpit Display System Interfaces to User Systems. Arinc Specification 661-2. 2004.
2. Basnyat, S, Chozos, N, Johnson, C, and Palanque, P. Redesigning an Interactive Safety-Critical System to Prevent an Accident from Reoccurring. 24th European Annual Conference on Human Decision Making and Manual Control. (EAM). 2005.
3. Basnyat, S., Chozos, N., & Palanque, P. (2005b) Multidisciplinary perspective on accident investigation. Special Edition of Elsevier's Reliability Engineering and System Safety Journal.
4. Cockton, G, Woolrych, A, Hall, L, and Hindmarch, M. Changing Analysts' Tunes: The Surprising Impact of a New Instrument for Usability Inspection Method Assessment. Proceedings of BCS HCI 2003. p. 145-162. 2003. Springer-Verlag.
5. Falla, M. Advances in safety-critical systems. Results and achievements from the DTI/EPSC R&D programme in safety critical systems. June 1997.
6. Gerhardt-Powals, J. (1996) Cognitive engineering principles for enhancing human - computer performance. International Journal of Human-Computer Interaction, 8(2), 189-211.
7. Hollnagel, E. (1991) The Phenotype of Erroneous Actions: Implications for HCI Design. IN: Human-Computer Interaction in Complex Systems (eds G. Weir & J. Alty), pp. 1-32. Academic Press.
8. Hollnagel, E. (1999) Accidents and barriers. IN: Proceedings of Lex Valenciennes (eds J.M. Hoc, P. Millot, E. Hollnagel, & P.C. Cacciabue), pp. 175-182.
9. Hollnagel, R and Woods, D. Epilogue: Resilience Engineering Precepts from of Resilience Engineering: Concepts and Precepts, Hollnagel, E., Woods D and Leveson, N (editors) 0 7546 4641 6. Ashgate. 2006.
10. Johnson, C. W. Failure in Safety-Critical Systems. A Handbook of Accident and Incident Reporting. Available at: <http://www.dcs.gla.ac.uk/johnson/book>. October 2003. University of Glasgow Press.
11. Killam, B and Holland, B. Position Paper on the Suitability to Task of Automated Utilities for Testing Web Accessibility Compliance [Electronic version]. The Usability SIG Newsletter: Usability Interface Accessibility and Usability: Partners in EffectiveDesign. 9, 4. April 2003.
12. Kletz, T. Hazop and Hazan: Identifying and Assessing Process Industrial Hazards. Institution of Chemical Engineers, Fourth Edition. December 1999. Taylor & Francis, Inc.
13. Leveson, N, Hunt, E. B, Jaffe, M, Joslyn, S, Rees, J, Shaw, A, Zabinsky, Z, Alfaro, L, Alvarado, C, Brown, M, Pinnel, D, Samarziya, J, Sandys, S, and Shafer, M. A demonstration of safety analysis of Air Traffic Control software: Final Report. September 1997.
14. 16. Leveson, N.G. (1991) Software safety in embedded computer systems. Communications of the ACM archive. ACM Press, 34, 34-46.
15. MDA Guide version 1.0.1. OMG Document number omg/2003-06-01. 2003.
16. Navarre, D., Palanque, P., & Bastide, R. (2003) A Tool-Supported Design Framework for Safety Critical Interactive Systems. Interacting with computers, 15/3, 309-328.
17. Navarre, D, Palanque, P, and Bastide, R. A Formal Description Technique for the Behavioural Description of Interactive Applications Compliant with ARINC 661 Specification. International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero'04). 2004.
18. Nielsen, J and Molich, R. Heuristic evaluation of user interfaces. Chew, J. C and Whiteside, J. Proceedings of Human Factors in Computing Systems, CHI' 90. 249-256. 1990. ACM.
19. Norman, D. The Design of everyday things. February 1990. Doubleday Books.
20. Palanque, P and Basnyat, S. Task Patterns for taking into account in an efficient and systematic way both standard and erroneous user behaviours. HESSD 2004 6th International Working Conference on Human Error, Safety and System Development. 2004.
21. Pierotti, D. Heuristic Evaluation - A System Checklist, Xerox Corporation Available online at <http://www.stcsig.org/usability/topics/articles/he-checklist.html>. 1995.
22. Puerta, A. R. Supporting User-Centered Design of Adaptive User Interfaces Via Interface Models. First Annual Workshop On Real-Time Intelligent User Interfaces For Decision Support And Information Visualization. 1998.
23. Pumfrey, D.J. (1999) The Principled Design of Computer System Safety Analyses. PhD Thesis. Univ. of York.
24. Rasmussen, J. Skills, rules, knowledge: Signals, signs, and symbols and other distinctions in human performance models. IEEE Transactions on Systems, Man, and Cybernetics. 13(3), 257-267. 1983.
25. Rushby, J, Crow, J, and Palmer, E. An automated method to detect mode confusions. 18th AIAA/IEEE Digital Systems Avionics Conference (DASC). 1999.

26. Schupp, B, Basnyat, S, Palanque, P, and Wright, P. A Barrier-Approach to Inform Model-Based Design of Safety-Critical Interactive Systems. 9th International Symposium of the ISSA Research Section Design process and human factors integration: 2006.
27. Smith, S. P and Harrison, M. D. Improving Hazard Classification through the Reuse of Descriptive Arguments. Gacek, C. 7th international Conference on Software Reuse: Methods, Techniques, and Tools. 2319, 255-268. 2002. LNCS Springer-Verlag.
28. 30. Sy, O, Bastide, R, Palanque, P, Duc-Hoa, KL, and Navarre, D. PetShop: a CASE Tool for the Petri Net Based Specification and Prototyping of CORBA Systems. 20th International Conference on Applications and Theory of Petri Nets. 1999. LNCS Springer Verlag.

COVER PAGE

Paper Title: Formal Socio-Technical Barrier Modelling for Safety-Critical Interactive Systems Design

Corresponding Author:

Sandra Basnyat
LIIHS - IRIT
UNIVERSITE PAUL SABATIER
118 route de Narbonne
31062 Toulouse Cedex 9
France
Email: basnyat@irit.fr
Phone: 00 33 561 55 74 04
Fax: 00 33 561 55 62 58
Site: <http://lihs.irit.fr/basnyat>

Summary:

This paper presents a three step approach to improve safety in the field of interactive systems. The approach combines, within a single framework, previous work in the field of barrier analysis and modelling, with model based design of interactive systems.

The approach first uses the Safety Modelling Language to specify safety barriers which could achieve risk reduction if implemented. The detailed mechanism by which these barriers behave is designed in the subsequent stage, using a Petri nets-based formal description technique called Interactive Cooperative Objects. One of the main characteristics of interactive systems is the fact that the user is deeply involved in the operation of such systems. This paper addresses this issue of user behaviour by modelling tasks and activities using the same notation as for the system side (both barriers and interactive system). The use of a formal modelling technique for the description of these three components makes it possible to compare, analyse and integrate them. The approach and the integration are presented on a mining case study. Two safety barriers are modelled as well as the relevant parts of the interactive system behaviour. Operators' tasks are also modelled. The paper then shows how the integration of barriers within the system model can prevent previously identified hazardous sequences of events from occurring, thus increasing the entire system safety.

Formal Socio-Technical Barrier Modelling for Safety-Critical Interactive Systems Design.

S.BASNYAT[†], B.SCHUPP[‡], P.PALANQUE[†], P.WRIGHT[‡]

[†]LIHS – IRIT, Université Paul Sabatier, 118, route de Narbonne, 31062 Toulouse Cedex 4, France

[‡] University of York, department of Computer Science, Heslington, York, United Kingdom.

Email: basnyat@irit.fr, bastiaan.schupp@jrc.it, palanque@irit.fr, peter.wright@cs.york.ac.uk

This paper presents a three step approach to improve safety in the field of interactive systems. The approach combines, within a single framework, previous work in the field of barrier analysis and modelling, with model based design of interactive systems.

The approach first uses the Safety Modelling Language to specify safety barriers which could achieve risk reduction if implemented. The detailed mechanism by which these barriers behave is designed in the subsequent stage, using a Petri nets-based formal description technique called Interactive Cooperative Objects. One of the main characteristics of interactive systems is the fact that the user is deeply involved in the operation of such systems. This paper addresses this issue of user behaviour by modelling tasks and activities using the same notation as for the system side (both barriers and interactive system). The use of a formal modelling technique for the description of these three components makes it possible to compare, analyse and integrate them. The approach and the integration are presented on a mining case study. Two safety barriers are modelled as well as the relevant parts of the interactive system behaviour. Operators' tasks are also modelled. The paper then shows how the integration of barriers within the system model can prevent previously identified hazardous sequences of events from occurring, thus increasing the entire system safety.

Keywords: Safety-Critical Interactive Systems, Barriers, Incident and Accident Investigation, Formal Specification Techniques, System Modelling, Human Factors

1. Introduction

Today, safety has become paramount in the design and operation of many technological systems. Often such systems present hazards that cannot be easily eliminated and therefore these systems become safety critical. Safety critical systems can be found in domains, such as in transportation, medicine, industry, and

even in financial systems. To mitigate the risk caused by the potential consequences of these hazards, risk reduction must occur for the system to be safe enough to be accepted by society. The means of risk reduction are usually dedicated safety systems that stop the evolution of scenarios leading to unacceptable consequences.

To avoid double use of the word system, we will refer to safety systems as safety barriers, or simply barriers. When we refer to 'the system', this reference is made to the system that is being designed and operated as a whole, for instance the plant, aircraft or computer system. Several definitions for barriers exist. For example,

“A barrier is an obstacle, an obstruction, or a hindrance that may either (i) prevent an action from being carried out or an event from taking place, or (ii) prevent or lessen the impact of the consequences, limiting the reach of the consequences or weakening them in some way” (Hollnagel 1999).

“The combination of technical, human and organizational measures that prevent or protect against an adverse effect” (Schupp et al. 2004).

“Barriers represent the diverse physical and organisational measures that are taken to prevent a target from being affected by a potential hazard” (Johnson 2003).

Though a long discussion of safety barriers is beyond the scope of this paper, barriers are usually regarded as systems that prevent or stop an undesired consequence. The ability to stop is important here, and defines the scope of what the barrier is. For instance a fire extinguisher is not a barrier itself, as it has to be operated by a human who must have received some training, and it must be in an easily accessible place. These elements are part of the barrier too. While systems and barriers should be independent to a certain aspect, they will often share components.

In this paper we will deal with a special but very common category of barriers, those that are socio-technical. This means that the barrier is essentially a combination of hardware and software, but also depends on human action for it to function correctly. The barrier thus assigns safety critical tasks to human operators who therefore become crucial in maintaining system safety. As these barriers are socio-technical, the tasks involve interaction with system software and hardware.

The task of the operator appears often hard to integrate in system design, and may occur too late (Daouk and Leveson 2001) whilst the technical part of designing a socio-technical system is often systematically

addressed and supported by notations and tools. These integration difficulties may lead to operators not being aware that a task is safety critical, which can obviously cause accidents.

It is the specification, analysis, verification and documentation of the safety critical human tasks that we are interested in. In this paper we outline an approach that facilitates these tasks. Most importantly, we simplify system analysis by explicitly defining barriers, analysing how these function, and only subsequently integrating them in the system, instead of directly trying to analyse the system as a whole.

2. The Approach

The approach employs a formal description technique to provide non-ambiguous, complete and concise models, thus giving an early verification of some potential problems to the designer before the application is actually implemented. However, formal specification of interactive systems often does not address the issues of erroneous user behaviour that may have serious consequences for the system. In order to provide such benefits, formal specification techniques can be complex, and designers may be reluctant to use them (Bowen and Hinchey 1999). For these reasons, the Interactive Cooperative Objects (ICO) approach presented in the paper is tool supported and tutorials, examples and case studies are available through the web site (<http://liihs.irit.fr/petshop>).

We use a three step approach (see Figure 1 for approach overview diagram). Step one uses the Safety Modelling Language (SML) to identify a structure which achieves risk reduction (Schupp et al. 2004). This structure makes up the safety architecture of the system. Here the specific hazards are analysed, and barriers are devised that can prevent targets (e.g. workers, environment) from being affected by these hazards. In this first step, barriers are treated as black boxes, it is specified why they are in the system, not how they function. Designers are thus supported in reasoning about risk reduction conceptually.

In the second step each individual barrier is analyzed, designed and modelled. Often various techniques are required to achieve this. In this paper we employ the Interactive Cooperative Objects (ICO) formalism (Bastide et al. 2000) based on Petri-nets to model and analyse the mechanisms of the barrier, their specifications and to verify their functions. The result of this step is a full design of the barrier which will achieve the safety function as specified in step one. This may use various parts of the system, hardware, software and human to achieve the required safety function.

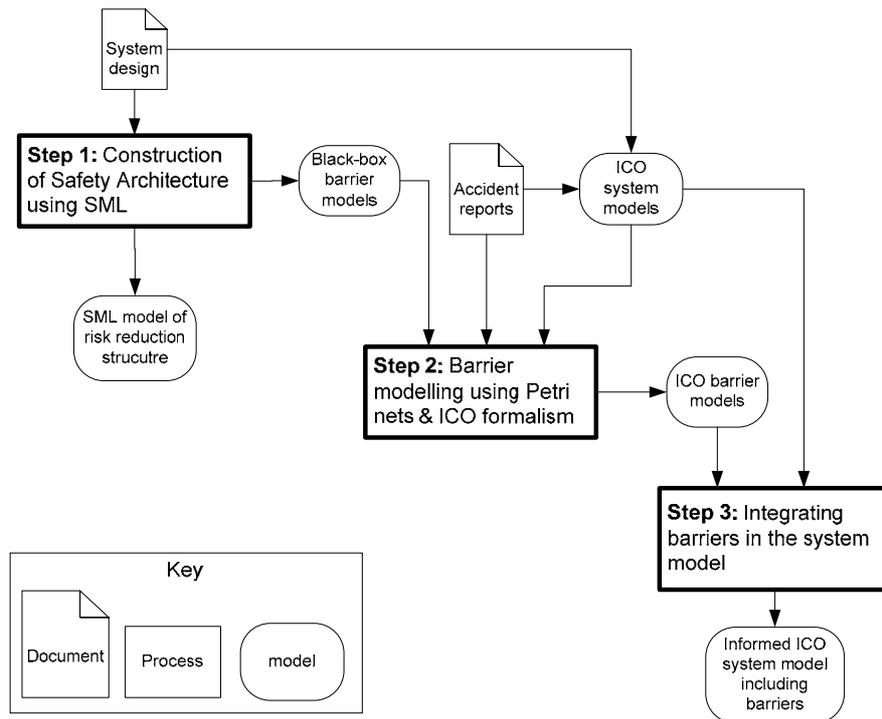


Figure 1. Approach Diagram

In the third step the functions specified by each individual barrier are connected to the system model as a whole by integrating the barrier into the system. In this paper we study this using the ICO-model. This occurs as follows. An operator has a number of functions. Some of these are specified by barriers. Using this mapping, it becomes clear which of the operator's functions are specified by which barrier, and therefore are safety critical. Barrier functions are connected in a similar way to hardware and software components of the system.

2.1 Safety Modelling Language

The first step of our approach uses Safety Modelling Language. In short SML (Schupp et al. 2001) uses the Hazard-Barrier-Target (H-B-T) model to model the safety architecture of a system. The H-B-T model assumes that targets are vulnerable to the effects of hazards, and that targets can be protected against these effects by barriers. In some respects it is similar to other barrier models, such as the accident evolution and barrier function model (Svenson 1991), and the 'Swiss-cheese' model (Reason 1990). In Figure 2, a basic example of a SML diagram is shown. It shows that toxic fumes are hazardous to workers. However the worker is protected by a containment system that contains the fumes, thus being a barrier that prevents

exposure. As this may not be completely adequate, the worker is further protected by Personal Protective Equipment (PPE). Alternatively prevention is realized by removing the hazard, for example by using a non-toxic substance.

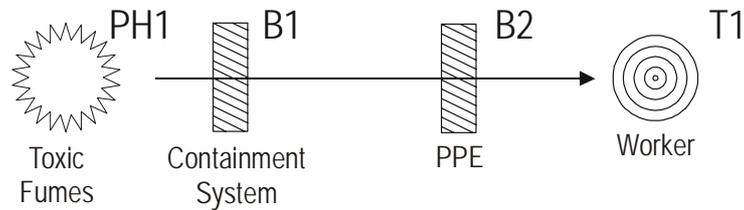


Figure 2. A typical H-B-T diagram. PH1 is a primary hazard symbol, T1 a target symbol, and B1 & B2 are barrier symbols

SML models hazards in a more complex manner than the basic H-B-T model in Figure 2. A hazard is something that has the potential to cause an adverse effect to a target. A hazard is a 'label' that humans apply to complex phenomena perceived as hazardous. SML does not provide insight into the hazardous phenomenon itself but into the relations this phenomenon has with the rest of the design/system. It is modelled using two components: Causal elements that provide a link to the mechanism of the hazard, and effects, that provide the link to the targets. For instance, when the elements 'flammable substance', 'oxygen', and 'ignition source' are present in a design, these will cause a fire hazard, having heat radiation, smoke and high temperature as effects. This is shown in Figure 3a. An example of a human factors related hazard is a misdiagnosis in interpreting an X-ray photograph in a medical domain. This can for instance be caused by the causal elements 'training', 'available time', and issues such as 'X-ray clarity'.

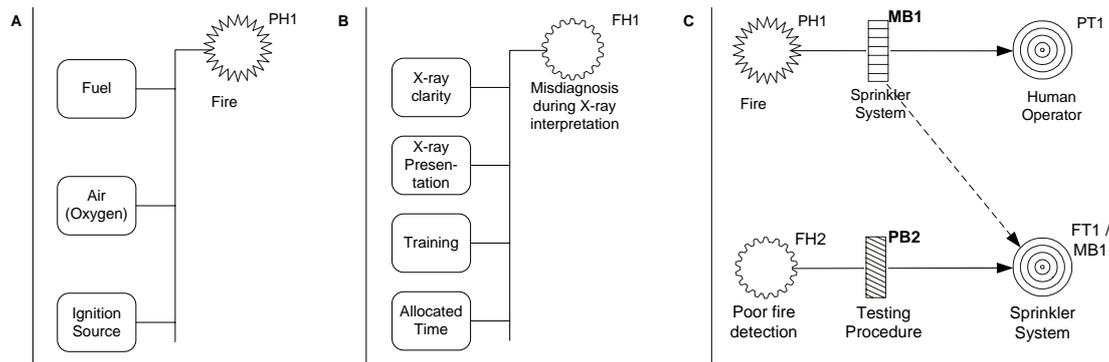


Figure 3: SML representation of (a) fire hazard, (b) a human error hazard, and (c) recursion. FH1 & FH2 are functional hazard symbols, MB1 a mitigative barrier, PB2 a protective barrier

To model the failure of barriers, SML defines primary and functional hazards. Primary hazards cause direct harm to humans, neighbouring installations, and the environment. The barriers inbetween primary hazards and primary targets are called primary barriers. Functional hazards are phenomena due to either human factors or other causes that adversely affect other barriers, thus making these fail. Poor fire detection causes a sprinkler system to become inoperable, a testing procedure protects against this, as shown in Figure 3c. In this way, a risk reduction problem is defined recursively; when a barrier is used, it can fail due to a functional hazard.

A consequence of this is that the list of primary hazards quickly provides insight into why the systems' safety is critical. Next, accident mechanisms, and the role humans play in these can be understood via recursions. This and many other aspects of the language such as the different kind of symbols and barriers are not further explained in this paper though Figure 3 shows some. For further information on SML see (Schupp et al. 2006).

2.2 System Modelling, Petri Nets, and the ICO formalism

Whilst SML helps to define the barriers and their role in risk reduction, we need to understand which tasks are defined by these barriers, and how they must be integrated in the system. SML is not helpful here. We use the ICO formalism based on Petri nets to achieve that. The ICO barrier models built using the SML model represents both human and system behaviour, thus allowing task analysis. The advantages of the use of formalisms are that they provide non-ambiguous, complete and concise notations. Moreover, they allow to check and prove properties of the design, thus to verify that the barrier will function.

2.2.1 Petri nets

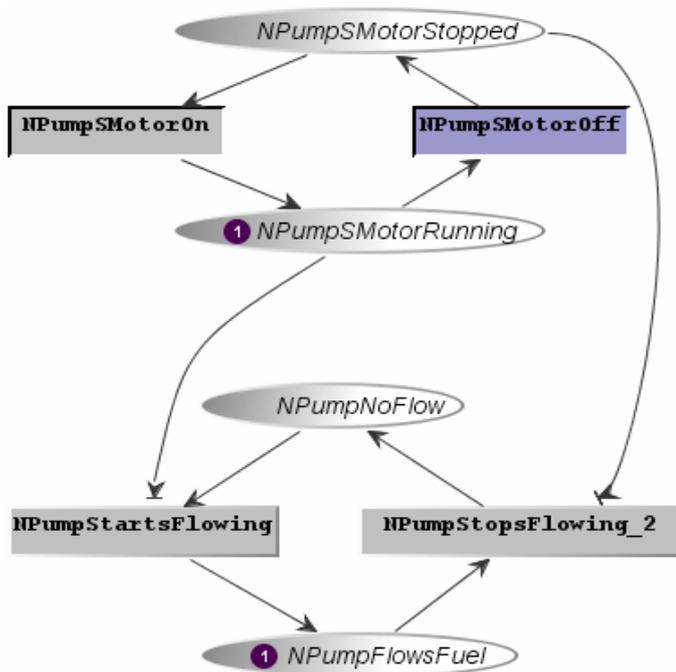
Petri nets are a widely used formal description technique in systems engineering. In this paper, a dialect of Petri nets is used to model both the system and the behaviour of barriers. Brevity prevents a detailed introduction to the Petri net notation however interested readers can look at (Petri 1962).

Petri nets are a formalism composed of four elements: the states (called places, depicted as ellipses), state changing operators (called transitions, depicted as rectangles), arcs (relating transitions and places) and tokens (representing the current state of the Petri net).

2.2.2 Informal presentation of the ICO formalism

The Interactive Cooperative Objects (ICOs) formalism is a formal description technique dedicated to the specification of interactive systems (Navarre et al. 2003). It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance, client/server relationship) to describe the structural or static aspects of systems, and uses high-level Petri nets (Genrich 1991) to describe their dynamic or behavioural aspects.

An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays information relevant to the user). An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented (Navarre et al. 2000). The specification can also be validated using analysis and proof tools developed within the Petri nets community. In subsequent sections, we use the symbols in Figure 4.



- States of the system are represented by the distribution of tokens into places 
- Actions triggered in an autonomous way by the system are called transitions and are represented as follows 
- Actions triggered by users are represented by half bordered transition 

Figure 4. An ICO model of the operation of a pump and its motor (see section 3 for full case study)

3. Case Study

The case study we have chosen to illustrate the proposed three-step approach is a fatal US mining accident (Andrews et al. 2002). We will first introduce the plant to the reader and then describe the accident. A Quarry and Plant system is designed to produce cement. However, the part we are interested in is the delivery of waste fuel used to heat the plant kilns. We will first detail the plant layout and its operation and then describe the accident. The Waste Fuel Delivery System is comprised of two separate liquid fuel delivery systems, the north and the south. Each system delivers fuel to the three plant kilns independently and cannot operate at the same time. See Figure 5 for layout diagram. This particular application was chosen because it relies upon the interactive operation of a complex, embedded command and control system involving human operators and technical systems.

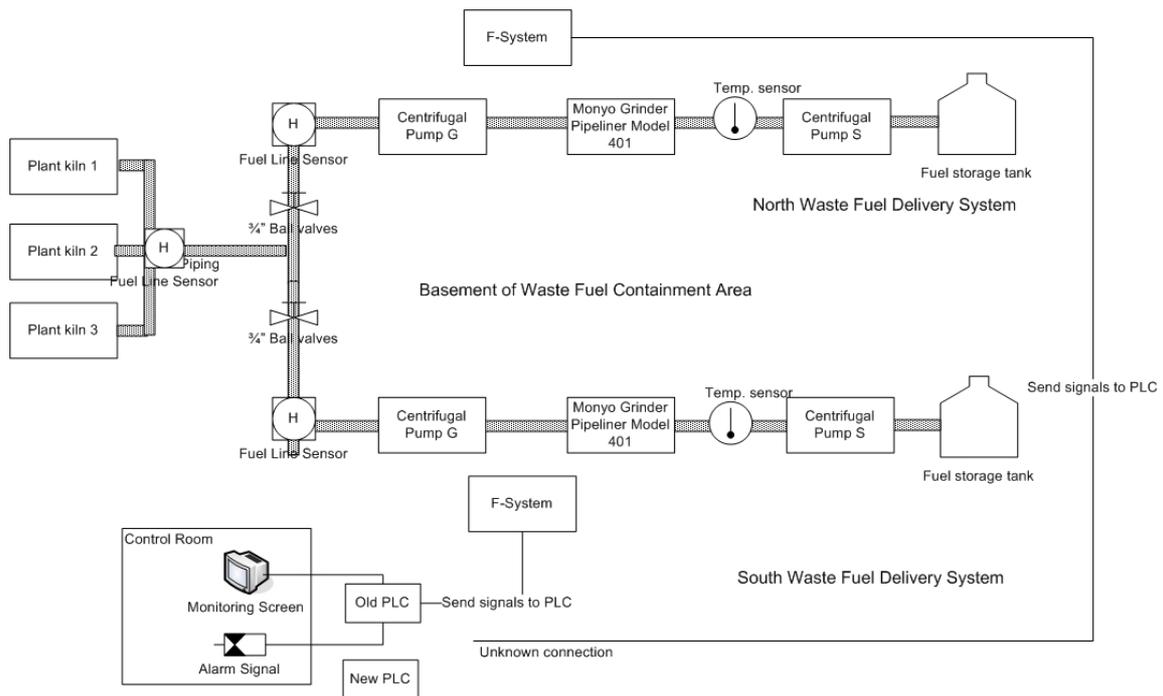


Figure 5. Simplified Layout Diagram of Waste Fuel Delivery Plant

Each delivery system contains the following components (presented here in the order of fuel flow):

- A fuel storage tank (right of the diagram)
- Two different sets of pumps, known as G and S, including motors and valves.
- A grinder, including motor
- One 3/4" ball valve

In order to understand the events leading to the accident, we must first describe the interaction between the command and control system and the north Waste Fuel Delivery System. If the north fuel storage tank is open, fuel flows from the storage tank to north pump-S. The north pump-S motor pumps the fuel to the north grinder. The fuel is grinded and passes to north pump-G. The north pump-G motor then pumps the fuel into the three kilns. The waste fuel delivery systems also contain sensors located in different areas. Each delivery system has at least one temperature sensor and one fuel line pressure sensor. There is also a pressure sensor in the plant kiln area where the north and south fuel lines join (H symbol in the diagram).

The sensors detect a number of abnormal conditions that will generate warning alarms in the control room. The command and control system will also automatically intervene to increase or decrease the speed of the pump motors when pressure is too low or too high. The fuel line pressure sensors also send information

directly to the pump-S and G motors to maintain the correct pressure of fuel to the three plant kilns via an automatic step increase program.

The waste fuel delivery system has two independent but interconnected electronic systems for monitoring and controlling both the north and the south fuel delivery systems. The 'F' system receives signals from sensors located on fuel lines. The data is transmitted to a programmable logic controller (PLC), which raises audible and visible alarms in the control room and can also update more detailed process information on the monitoring screen.

3.1 Personnel

A number of different individuals contributed to the events that led to this accident. A Kiln Control Operator is located in the control room. Her responsibility is to monitor the fuel system via the screens and respond to any alarms. If necessary, an emergency shutdown of all pumps can be activated from the control room. The Kiln Control Operator uses a two-way radio to interact with other personnel located in the basement of the waste fuel containment area. The victim of the accident was responsible for monitoring the Waste Fuel Delivery Systems and for responding to instructions given by the supervisor who was located in the containment area. This supervisor monitors the Waste Fuel Delivery Systems and the worker. The supervisor communicates with the Kiln Control Operator via two-way radio. He receives information about the fuel flow from the Kiln Control Operator. The supervisor instructs the worker to make adjustments to the systems and switch the active Waste Fuel Delivery System from north to south or vice versa. The supervisor also assists the worker, if needed.

3.2 Current barriers imposed in the plant

During the accident analysis, a number of barriers were identified as being present within the system design.

3.2.1 Hardware related barriers

Line sensors and alarms: The control room monitored the two systems by means of sensors that activated alarms and displayed warnings on monitor screens when specific set points for temperature or pressure were exceeded. Three pressure sensors were installed on the fuel line: one in the kiln area of the plant, and one each on the north system and south system just prior to the point where the two system lines merged into a single

line leading to the plant. If an alarm sounded or a warning was displayed, the control room operator used two-way radios to communicate with personnel at other areas.

Containment area: Contains hazardous fuel from humans and igniting sources.

Auto-shutdown: Two independent but interconnected electronic systems monitored and controlled both the north and south fuel delivery systems. The “F System I/A DCS”, (Intelligent Automation, Distribution Control System), is used to monitor and record normal operating parameters (temperatures, pressures, etc.) as well as audible and visual alarms. These can be viewed on displays at the plant control room. A PLC (Programmable Logic Controller) network performed the basic start-up/shutdown of the system and responded to electronic commands from the F System. The F System recorded information it sensed, but it did not record the PLC's actions.

One of the commands that the F System is programmed for, is send a signal to the PLC to de-energize all pumps in the fuel delivery system if a pressure of approximately 60 pounds per square inch (PSI) was not sensed at the line in the kiln floor area within 3 minutes of system start-up. The 3-minute set point was based on a normal delay of 3 minutes for pressure to reach approximately 60 PSI at the kiln area from the time the pumps were started.

The fire suppression system: Located in the waste fuel containment area and should activate after a fire breaks out.

Manual shut off valves: Implemented and used to prevent the north system from operating at the same time as the south system.

3.2.2 Human related barriers

Manufacturer guidelines and procedures: Among many guidelines, the manufacturers strongly recommend against bleeding pipes while pumps are in operation

Training: The operators had both attended training classes on the operation of the waste fuel system.

3.3 Events Leading to the Accident

A seal on the north grinder overheated. The Kiln Control Operator and supervisor decided to switch waste fuel delivery systems from north to south. The worker switched delivery systems; however fuel did not flow to the plant kilns as planned. The personnel believed the problem was due to air being trapped in the south

fuel pipes. They, therefore, bled the valves of the south system while the motors were running. In the meantime, due to the low pressure being sensed in the fuel lines, the automatic step increase program was increasing the speed of the motors on the south pumps in an attempt to increase pressure in the fuel line. These various factors combined to create a ‘fuel hammer effect’ in the pipe feeding the south pump. The hammer effect is caused by rebound waves created in a pipe full of liquid when the valve is closed too quickly. The waves of pressure converged on the south grinder. Figure 6 provides an overview of this ‘fuel hammer effect’. The grinder failed and fuel was sprayed on the two personnel. The fuel ignited and fire spread across the entire area. The supervisor managed to extinguish himself only. The worker later died in hospital.

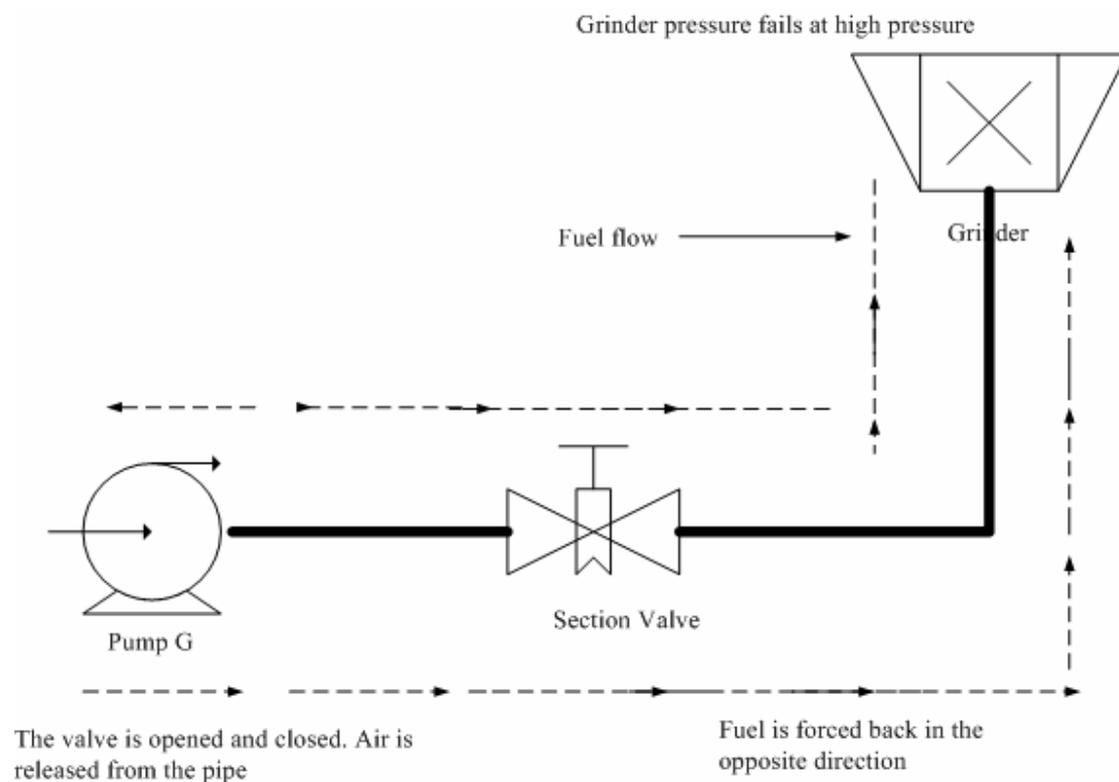


Figure 6. Fuel Hammer Effect

We will focus at the start-up procedure of the fuel line. We will analyse why some of these tasks should have been defined as safety critical, and how our method is of help here. We will investigate how using barriers improves design by helping to define safety critical operator tasks. Lastly we will discuss the integration of these barriers with the system.

3.4 System Analysis, starting point

In a real world design situation a natural starting point for our analysis would exist, provided by drawings or other documentation of the design. We therefore shortly discuss the design of the fuel line before proceeding with explaining the proposed method. At this point, hazards and barriers are not yet known, hence we start from a purely a functional description. We will discuss a hazard analysis as well, though this is not strictly part of our method.

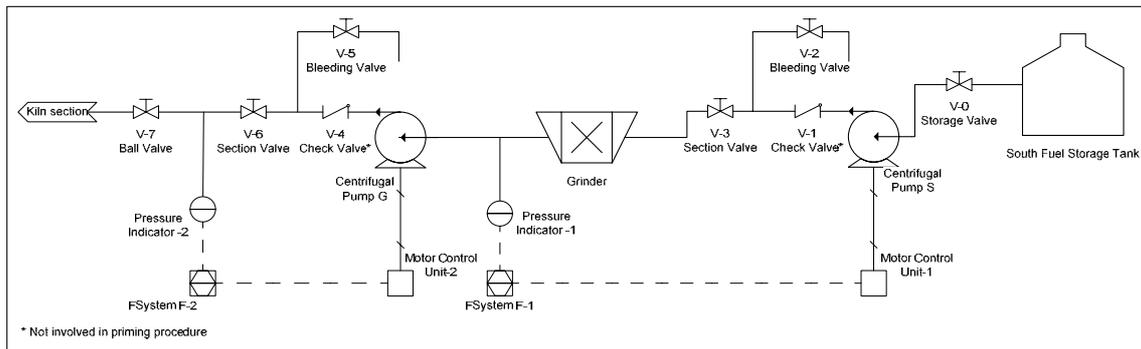


Figure 7: Piping and Instrumentation Diagram of fuel line. See text for explanation.

The exact manner in which the design is represented is domain dependent. Here we use a Piping and Instrumentation Diagram (P&ID), and a formal model. A formal system model is a required input to our method. Conversely, the P&ID is only required to understand the design, and may be substituted or complemented by other forms of documentation. In a real world situation, the formal model may have to be created before proceeding.

3.4.1 Functional system design

Figure 5 shows the Piping and Instrumentation Diagram (P&ID) describing the Fuel delivery system. V-1 and V-4 represent check valves, V-2 and V-6 are section valves, V-3 and V-5 are for bleeding, and V-7 is a ball valve used to control the start-up procedure. Bleeding is done before system start-up to make the pipes free of air, and to prime the pumps. Without priming, the pumps cannot create suction, and thus do not pump. MCU-1&2 are the Motor Control Units, PI-1&2 are Pressure Indicators, FSystem-1&2 are the implementations of the pump control loops.

3.4.2 Formal system description

The formal system description is achieved using the ICO formalism described in section 2.2. The model is here only briefly discussed, see [12] for a more in depth discussion.

In [12], we have modelled each individual component of the plant (e.g. pumps, grinders, fuel tanks etc) using the ICO formalism. These individual components have been interconnected based on fuel flow, because that most accurately describes how the process functions. With this configuration, we are able to see what happens to fuel if for example a motor is not turned on, or a valve is left open etc. Though Figure 8 is illegible, it gives an overview of the complete system model for the fuel system. For explanatory purposes, the components have been grouped and labelled.

- | | |
|--|---|
| A) Fuel tank | F) Pump-G Motor and corresponding fuel flow |
| B) Pump-S Motor and corresponding fuel flow | G) Pump-G Bleeding Valve (V5 in Figure 5) |
| C) Pump-S Bleeding Valve (V0 in Figure 5) | H) Pump-G Section Valve (V6 in Figure 5) |
| D) Pump-S Section Valve (V3 in Figure 5) | I) Ball valve (V7 in Figure 5) |
| E) Grinder Motor and corresponding fuel flow | J) Plant Kilns |

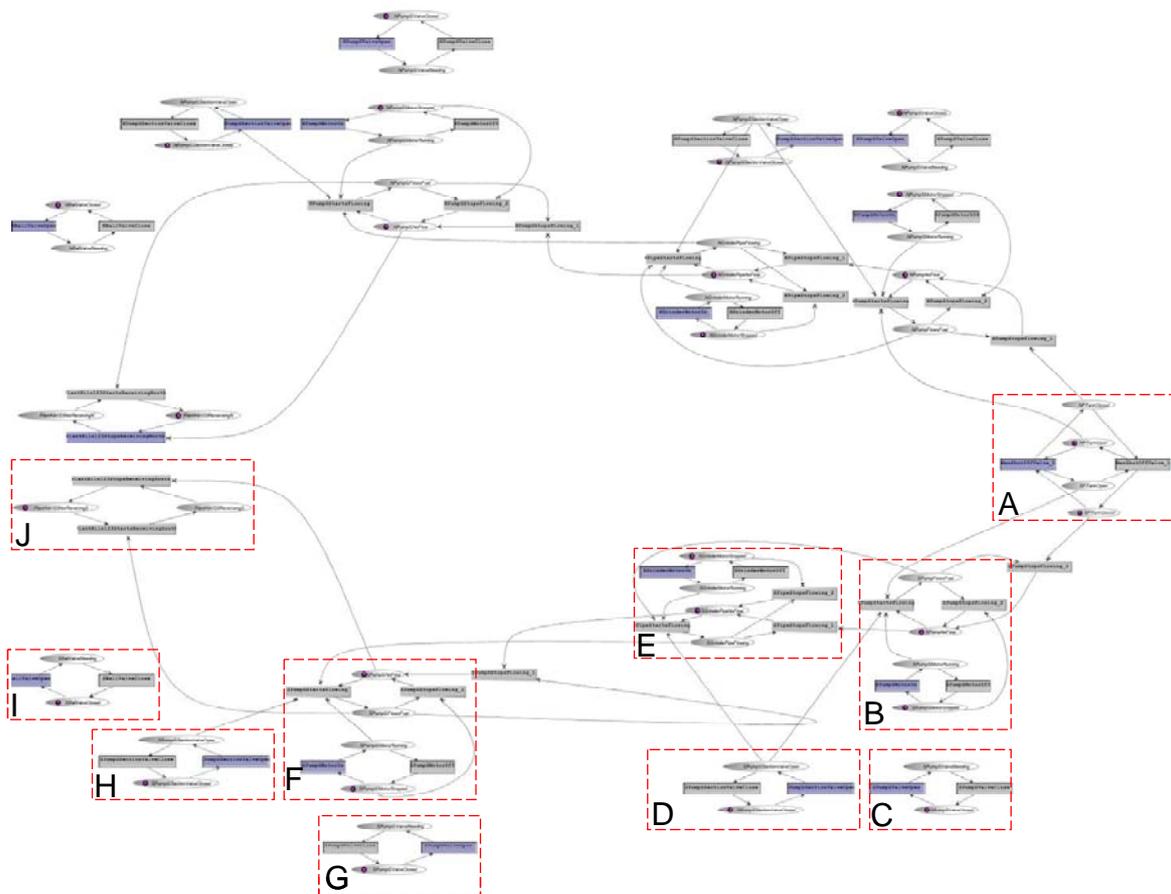


Figure 8. Petri net system model of the fuel line

4. Application of the approach

This section describes the application of our three-step approach on the case study.

4.1 Step 1: HBT analysis

In this example hazard identification is relatively straightforward. A group of experts using a common identification method (e.g. a Hazop) would for instance quickly realise that both too much static pressure and pressure waves (which may compromise containment) can occur in this system. Though other hazards might be identified as well, we will concentrate on these pressure related hazards here.

Our method however becomes important after hazard identification. Now two steps must be taken. Firstly, potential consequences and risk must be estimated, then barriers to reduce the risk must be designed. Though both steps are supported by the method, this paper focuses at designing the barriers, and in particular at integrating and understanding tasks carried out by humans as part of these barriers. In this example, typical barriers may include surge arrestors, emergency shutdown or procedural barriers (e.g. limiting pump power during start-up). The method helps to select, specify and verify these.

In Figure 9 the results of a Hazard-Barrier-Target analysis using SML are shown. The primary hazard is a fire hazard, as this will cause harm to for instance, operators. Many potential barriers are available that may stop them from receiving such harm. Fire normally is caused by the presence of fuel, air and an ignition source. In this case fire is prevented by an Inherent Barrier (IB1), containment, which keeps the fuel separated from air and ignition sources. In case containment (IB1) fails, and fire occurs, a sprinkler system is present to mitigate the effects of the fire. Figure 9A shows a SML representation of this. More complicated and precise models are probably appropriate here, for instance to include failure modes of the barrier (e.g. spraying fuel or just leaking). SML facilitates these, but for brevity we will omit further discussion.

Containment thus is an important barrier here. Our further discussion focuses on protecting this barrier against functional hazards. That is, the integrity of pipe work and casing of equipment such as the pumps and the grinder must remain. This may however be compromised by high pressure in the system. Therefore pressure surges must not occur in the system. In other words, these are functional hazards. A pressure surge may for instance occur because of starting a pump in the wrong way. For this to happen, four causal elements must be present; the pump must be running, but it must contain air as is shown in Figure 9B. Then the air must be bled from it, which will cause sudden flow. In an inelastic system, this will cause a pressure surge.

Hence, bleeding the pump (also called priming; fill it with liquid as it cannot otherwise create suction), and starting the pump must not occur in the wrong order. Two alternative causes of high pressure are possible as well; water hammer due to sudden stop of flow, and static high pressure, see Figure 9C.

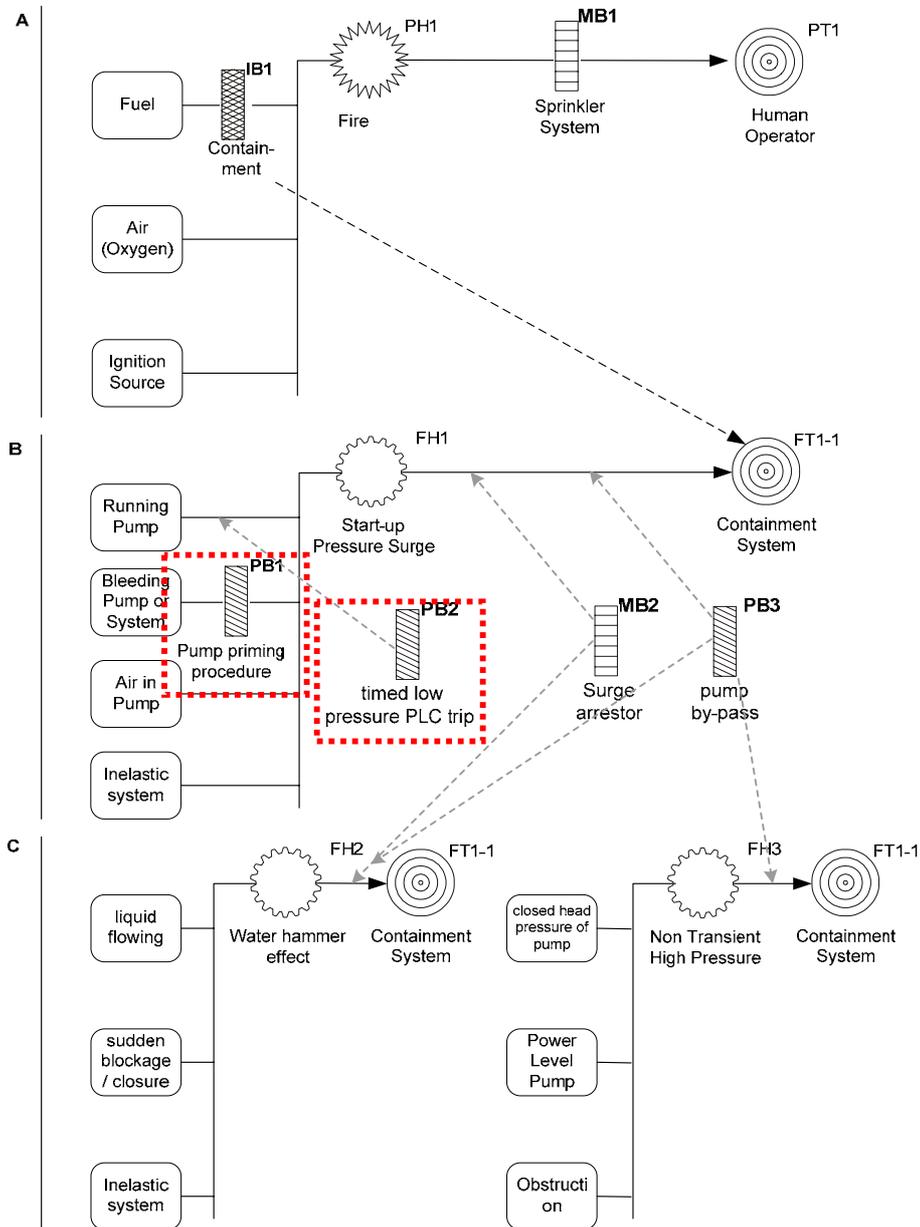


Figure 9. SML representation of hazards, barriers and targets discussed in this text (PB1 and PB2 modelled in this paper)

The system designers may now evaluate alternative barriers that potentially reduce the risk of a pressure surge. The SML notation allows system designers to conceptually understand which barrier is best to use. Although PB1, the pump priming procedure, will prevent pressure surges (FH1) altogether, MB2 will protect against most transient pressure waves (FH2), now also including water hammers. PB3 will protect containment against both transient (FH1 and FH2) and non-transient (static) high pressure (FH3), as the dotted arrows indicate.

In deciding, designers also have to take into account factors like probability of failure on demand, economic viability, and ease of design, construction and operation. This is not subject of further discussion here. As we are interested in barriers, and in particular, socio-technical barriers, we continue building an ICO model of PB1 and PB2.

4.2 Step 2: Barrier modelling using Petri Nets & ICO formalism

In this section we discuss the analysis and design of PB1 and PB2. PB1 has been selected for modelling because it is clearly a socio-technical barrier. It is an operator procedure, derived from manufacturer guidelines involving human operators and their interactions with the system. PB2 will also be modelled however can be considered a technical barrier since it involves valves, a programmable logic controller and the “F-system”. In this step the focus thus is on the barrier, not on the system as a whole. In our view, this is an important improvement over current design practices as the design of barriers and systems often becomes intermingled, causing people to loose track of which elements of the system actually are part of barriers.

4.2.1 ICO modelling of PB1: Pump Priming Procedure

As discussed a pump in this system should not run dry or be started unless it has been sufficiently "primed". Before describing the model of this procedural barrier, we first present informally the priming process exemplified on the fuel delivery system (that is graphically presented in Figure 7).

The principle of this procedural barrier is simple. First prime the pump, then switch it on. As the system contains two pumps and some other components, the actual barrier is more complex. A domain expert might design it as follows:

1. Before and during the priming procedure, no motor must be on (i.e. both pump motors and grinder motor)
2. Open the fuel storage tank by opening V0 Storage Valve. (Assumption: gravity will cause fuel to flow through the piping until V3 Section Valve)
3. Open V2 Bleeding Valve to release any air in piping.
4. When all air is removed, close V2 Bleeding Valve
5. Open V3 Section Valve. (Assumption: gravity will cause fuel to continue flowing to the next Section Valve V6).
6. Open V5 Bleeding Valve to release any air in piping.
7. When all air is removed, close V5 Bleeding Valve
8. Open V6 Section Valve. (Assumption: gravity will cause fuel to continue flowing to the kiln section).
9. If required, close V3 and V6 Section Valves (when the system is not to be used immediately).

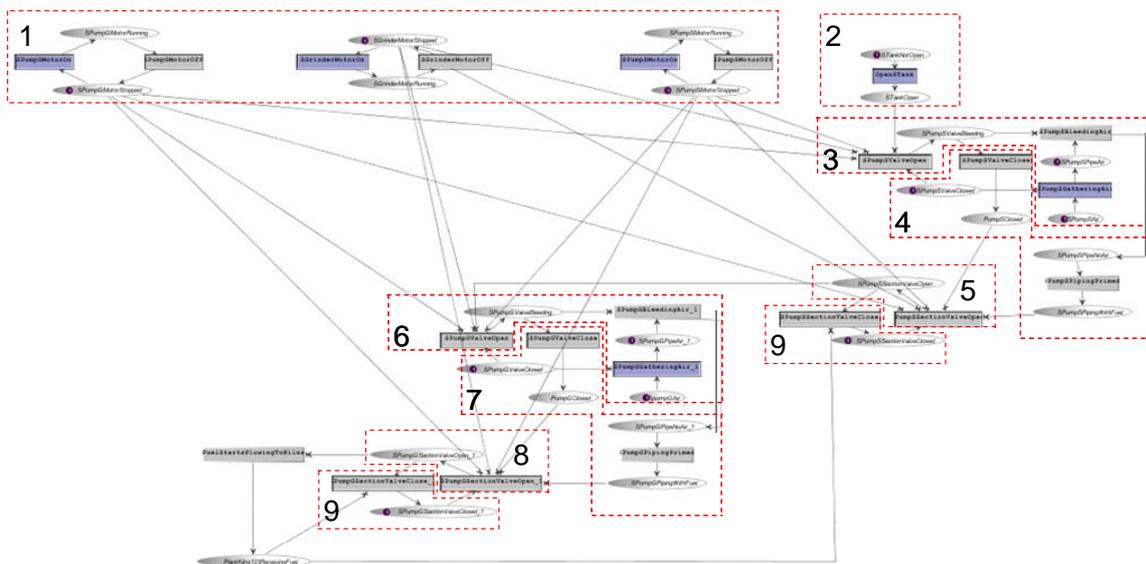


Figure 10. PB1 Pipe Priming Procedure Barrier

Notice that this barrier reuses some functionality already present in the system. For instance, the pumps do not pump without being primed, therefore priming is part of the functionality with or without this barrier. The PB1 barrier imposes constraints on this existing functionality to ensure safety. This becomes explicit by

modelling the barrier this way. At the same time the barrier also defines new functionality, and implements it in other system components. For instance, the first step demands that the pump motors must remain off until the procedure defined by PB1 is completed. This is simple to implement in the barrier model, however in practice this can be more complicated, and detailed discussion of this is beyond the scope of our discussion. It may perhaps be achieved using signs near the switches referring to the procedure.

We have modelled PB1 using the ICO formalism - see Figure 10. The diagram has been segregated into several sections for explanatory purposes. The model of this barrier is a combination of hardware and human actions. The hardware concerns the three motors, modelled in Figure 10 part 1, the Pump-G motor, Grinder motor and Pump-S motor. It also concerns the valves. However these are modelled together with their interaction with the required operator's actions in the remainder of the figure.

The barrier is mainly made up of arcs connecting transitions and places rather than the transitions and places themselves. It can be decomposed into three main parts. The first part concerns preventing the three motors (part 1 of Figure 10) from running before the procedure is completed. This is modelled by means of test arcs which impose a pre-condition on operating the bleeding valves (V2 and V5) or the section valves (V3 and V6) preventing them from being opened during the priming procedure if the motors are running.

The second part of PB1 is the obligation of order of events for the operator's interaction with the system. For instance part 5 shows the opening of the 1st section valve which cannot be opened before air has been bled from the first section of piping (part 3).

The third part is also an obligation of order. It prevents the operator from closing the section valves (as shown in part 9) before fuel arrives at the plant kilns (a token is set into place `PlantKiln123ReceivingFuel`). Once fuel arrives at the plant kilns, the task is complete. The closing of the section valves (step 9 in the procedure) is optional and depends on whether the fuel delivery system will be started immediately in which case the section valves are left open, or later, in which case they can both be closed.

4.2.2 ICO modelling of PB2 Auto-Shutdown of Pumps

The second Primary Barrier (PB2) has also been modelled using Petri nets and the ICO formalism. Figure 11 illustrates the Petri net representing this barrier. It is interesting to that it is very different from the priming procedure barrier previously described.

As previously mentioned this is also one of the barriers designed and present in the existing mining plant. One of the commands that the F-System was programmed to send to the PLC, was to de-energize all pumps in the fuel delivery system if a pressure of approximately 60 pounds per square inch (PSI) was not sensed at the line in the kiln floor area within 3 minutes of system start-up. The 3-minute set point was based on a normal delay of 3 minutes for pressure to reach approximately 60PSI at the kiln area from the time the pumps were started. The accident occurred 10 minutes after the pumps started. When no pressure was detected in the line at the kiln area, the F-System functioned properly in signalling the PLC to shut down the pumps 7 minutes before the accident, but the PLC did not respond. It failed the day of the accident because three months prior to the accident, an older PLC system was replaced by a new PLC, and the F-System connections to the older PLC system were never changed over to the new PLC.

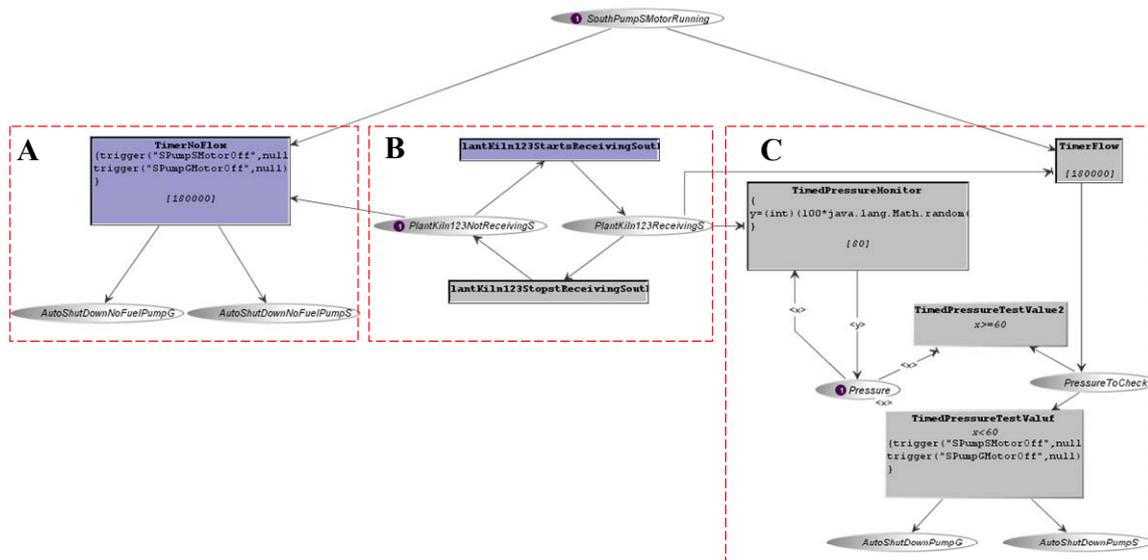


Figure 11. PB2: Auto shut down of pumps after 3 minutes barrier

Figure 11, diagram of PB2, has been separated into three sections for explanatory purposes. In order to model this barrier, we have assumed that the F-System system starts counting to 3 minutes from the moment the South Pump-S Motor is started (the first motor in line from the fuel tank according to the order of fuel flow). The place at the top of the diagram containing a token, SouthPumpSMotorRunning, represents the behaviour that the motor is running. With respect to the barrier, there are main two possibilities at the end of the 3 minutes; either no fuel arrived at the plant kilns (modelled in parts A and B) or fuel arrived at the plant kilns (modelling in parts C and B). Part B of the diagram represents the plant kilns. Fuel flows to the plant

kilns via the north or the south waste fuel system. The component in this diagram represents the kilns receiving or not receiving fuel, from the south delivery system only. The initial marking (represented by setting a token in the place PlantKiln123NotReceivingS) of the kilns in this model shows that the kilns are not receiving fuel from the south delivery system.

In the first scenario, where no fuel arrives at the kilns by 3 minutes, the TimerNoFlow transition which has a timer of 180,000ms will fire, remove the token from SouthPumpSMotorRunning and from PlantKiln123NotReceivingS and set a token in AutoShutDownNoFuelPumpG and AutoShutDownNoFuelPumpS.

In the second scenario, where fuel does arrive at the kilns by 3 minutes, the auto shut-down of the pumps depends on the pressure of the fuel in the pipes. If the pressure is less inferior to 60PSI, the pumps must be shut down. To simulate the scenario in which fuel arrives, the PlantKilns123StartReceivingSouth transition can be fired which will remove the token from PlantKiln123NotReceivingS and set a token in PlantKiln132ReceivingS. The moment fuel begins to arrive at the kilns, the transition TimedMonitorPressure sets a token in the Pressure place and gives it random values (representing the PSI pressure) every 80ms (in this model).

In the top right hand corner of part C of the diagram, we have another timer (the same as in part A) lasting for 180,000ms. After 3 minutes, this transition will fire removing the token from SouthPumpSMotorRunning (Note, the token containing a random value is not taken from PlantKiln132ReceivingS because fuel continues to arrive. This is modelled using a 'test arc'). The timer transition sets a token in the PressureToCheck place. From this state, two further transitions will become fireable, depending on the random value that the token in the PressureToCheck place has been set to. If the value of the token is superior to 60, the upper transition, TimePressureTestValue2 will fire automatically, and the system will simply continue to function. If on the other hand, the random value of the token is inferior to 60, the lower transition TimePressureTestValue will fire and set a token in AutoShutDownPumpG and AutoShutDownPumpS.

It must be noted however, that the barrier model of PB2 could be considered as incomplete because it currently does not shutdown the pumps. The complete model of the barrier includes events and arcs connected to the pump motors in order to represent the connections between the models and thus automatically turn pumps G and S off.

Now the barrier models have been established they can be analysed further. For instance human factors methods can be used to understand whether humans can achieve the tasks that the barriers specify.

4.3 Step 3: Connecting technical and human barriers in the system model

Integrating the barrier in the system in the real world obviously involves much more than just integrating the ICO models. However, we will continue our discussion on the integration of barriers with the system model. Figure 12 illustrates steps 2 and 3 of the approach. Step 2 (described in the previous section) of the diagram details three ways in which a barrier can be modelled. The barrier could simply correspond to adding arcs between existing components of the system model. However, it is more likely that the barrier will comprise of one or more of the following; new arcs between existing components of the system model, new components, and new arcs between existing components of the system model and new components. The modelled barrier could only have an impact on a subset of the system model but it is naturally difficult to model a barrier without exploiting the system model. Step 3 (described in this section) is the integration of the barrier model with the system model.

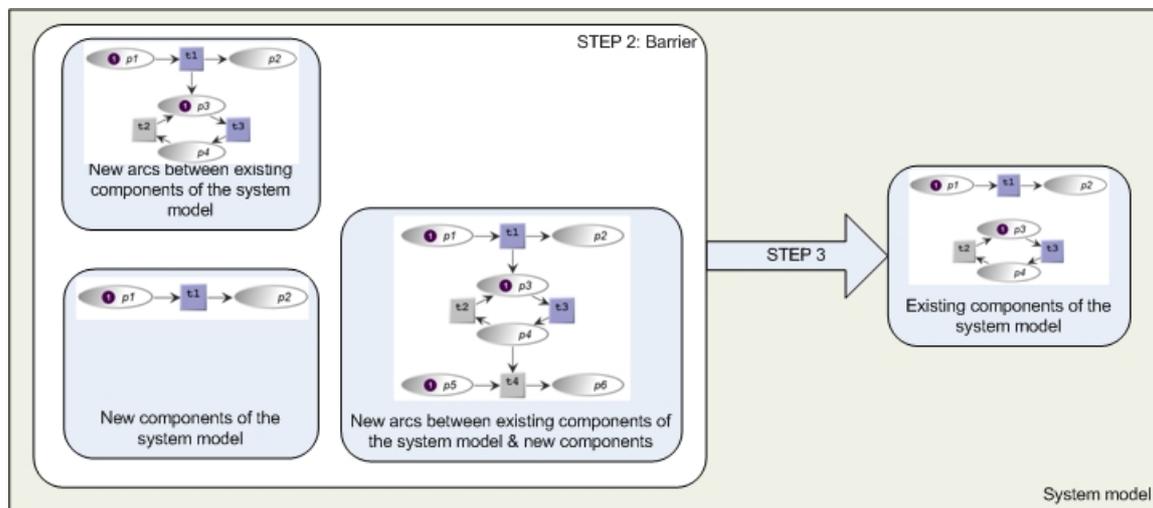


Figure 12 Steps 2 and 3 of the approach – barrier modelling and integration.

As discussed, the initial state required by PB1 might be implemented by adding a sign to a power switch on pump, which refers to the procedure specified by PB1. Such implementation issues are beyond the scope of this paper however, and require additional methods. Here we only have space to briefly discuss what happens to the ICO models.

The initial ICO system model as presented in Figure 8 significantly changes because of the connection of PB1. In Figure 13, we present the system model (previously shown in Figure 8) with the addition of PB1 in the south waste fuel delivery system. Although illegible, the idea is to illustrate the complexity of adding a simple procedural barrier. The number of arcs has significantly increased.

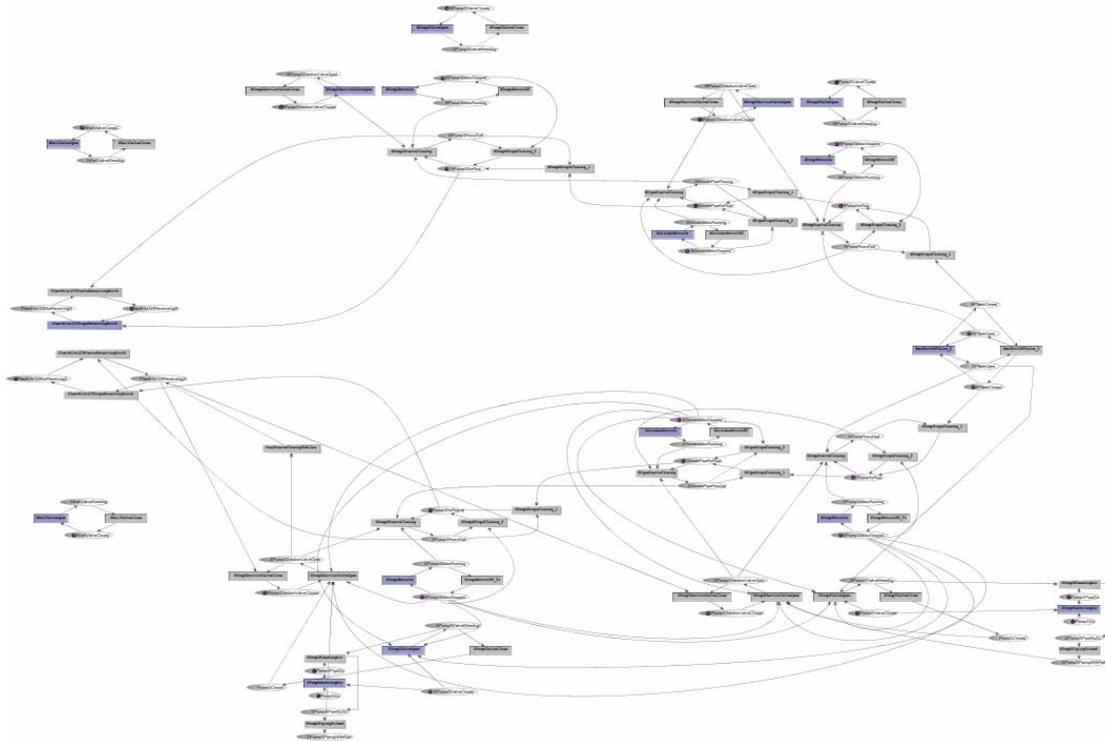


Figure 13. System model + PB1

The complexity of the model has also significantly increased, mainly because of the number of added arcs. Also several places and transitions have been added for instance to represent the existence of air in the pipes which was previously not represented in the system model as this has nothing to do with the initial functional specification of the system. Although out of the scope of this paper, the issues of model illegibility due to the complex nature of the system under design, have recently been tackled. The tool used for ICO modelling, Petshop, has been extended to make it usable for real size interactive applications. Briefly, the three extensions are *Restructuring models*, *label hiding* and *virtual places*. The interested reader can see (Barboni et al. 2006)

Two further issues must be noted. Firstly the addition of the barrier to the system model must not change the behaviour of the system, except of course for excluding the hazardous state it is designed for. That is to

say, an action or task that was previously available on the system side or via interaction with the user must not be changed and must be still available. Secondly, the actual modelling of the barrier inevitably involves direct integration with the system model because the barrier not only takes existing components, but may also rely on adding extra arcs to existing components. Hence it is not possible to model the barrier without taking notion of the system model; barriers exploit a subset of the system model, sometimes with extra transitions and places. Thus step 3 of the approach is the addition of the complete barrier, including any additional components that were not present in the existing system model.

Using the ICO model representing the system behaviour, it becomes possible to verify that the system still works, to prove that the hazardous state is no longer reachable and to analyse in which way the newly integrated barrier may fail. Briefly, an ICO specification can be executed to provide a prototype User Interface (UI) of the application under design. It is then possible to proceed with validation of the system by applying techniques such as user testing. In (Basnyat et al. 2005a) we present a systematic approach for exploring all possible combinations of actions and events of an ICO system model using typical Petri net analysis techniques such as marking graphs, to analyse our model and ensure that a hazardous state is no longer reachable. That is to say, that a place, representing a hazardous state can no longer contain a token.

5. Conclusions and future work

This paper presented a three-step approach for identifying, modelling and formally specifying safety critical human tasks, interactive system and their associated barriers. The Safety Modelling Language is used to model existing as well as identify new socio-technical barriers. We then used the Interactive Cooperative Objects (ICO) formalism to specify the behaviour of the barriers. Finally we integrate these barriers with a system model, also specified using the ICO formalism. To date we have successfully integrated individual barriers with the system model for simulation purposes. However, it is important to note that integration of multiple barriers raise additional issues related first to the increase of the size of the models, second to the potential conflicts between the various barriers. However, we believe that formal description techniques can provide a unique way of detecting such conflicts even though their resolution remains an open issue.

The approach presented is part of a larger framework of research centred on model based design, aiming to improve the design of safety critical interactive systems by accounting for errors (technical and human

related) early in the design process. We believe that by identifying and incorporating socio-technical barriers such as those discussed in this paper within their relevant models, we can obtain an early verification of some potential problems before the application is actually implemented. This will ultimately lead the design of safer safety critical interactive systems by embedding reliability, efficiency and error-tolerance in the system. For instance, as part of this larger framework, we have shown in previous papers (Basnyat et al. 2005b) how system design can be improved by using accident investigation techniques.

6. Acknowledgements

This work was supported by the EU funded ADVISES Research Training Network, GR/N 006R02527. <http://www.cs.york.ac.uk/hci/ADVISES/>

7. References

- [1] Andrews, W. S, Anguiano, J. Y, Palmer, B. B, and Skrabak, R. A. Exploding Vessels Under Pressure Accident. Mine I.D. No. 03-00256. October 24, 2002.
- [2] Barboni, E, Navarre, D, Palanque, P, and Basnyat, S. Addressing Issues Raised by the Exploitation of Formal Specification Techniques for Interactive Cockpit Applications. International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero). 2006.
- [3] Basnyat, S, Chozos, N, Johnson, C, and Palanque, P. Redesigning an Interactive Safety-Critical System to Prevent an Accident from Reoccurring. 24th European Annual Conference on Human Decision Making and Manual Control. (EAM) Organised by the Institute of Communication and Computer Systems. 2005.
- [4] Basnyat, S., Chozos, N., & Palanque, P. (2005b) Multidisciplinary perspective on accident investigation. Special Edition of Elsevier's Reliability Engineering and System Safety Journal.
- [5] Bastide, R, Sy, O, Palanque, P, and Navarre, D. Formal specification of CORBA services: experience and lessons learned. ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'2000). 2000. ACM Press.
- [6] Bowen, J. P and Hinchey, M. G. High Integrity System Specification and Design. 1999. London, Springer-Verlag.
- [7] Daouk, M and Leveson, N G. An Approach to Human-Centered Design. Workshop on human error and system development. <http://sunnyday.mit.edu/papers.html> . 2001.
- [8] Genrich, H. J. Predicate/Transitions Nets, High-Levels Petri-Nets: Theory and Application. K. Jensen & G. Rozenberg (eds.). High-Level Petri: Theory and Applications. 3-43. 1991. Springer Verlag.
- [9] Hollnagel, E. (1999) Accidents and barriers. IN: Proceedings of Lex Valenciennes (eds J.M. Hoc, P. Millot, E. Hollnagel, & P.C. Cacciabue), pp. 175-182. Presses Universitaires de Valenciennes.
- [10] Johnson, C. W. Failure in Safety-Critical Systems. A Handbook of Accident and Incident Reporting. Available online at: <http://www.dcs.gla.ac.uk/johnson/book>. October 2003. Glasgow, Scotland, University of Glasgow Press.
- [11] Navarre, D., Palanque, P., & Bastide, R. (2003) A Tool-Supported Design Framework for Safety Critical Interactive Systems. Interacting with computers, 15/3, 309-328.
- [12] Navarre, D, Palanque, P, Bastide, R, and Sy, O. Structuring Interactive Systems Specifications for Executability and Prototypability. 7th Eurographics Workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'2000. no, 1946, 97-109. 2000. Lecture Notes in Computer Science, Springer.
- [13] Petri, C. A. Kommunikation mit automaten. PhD Thesis. 1962. Technical University Darmstadt .
- [14] Reason, J. Human Error. 1990. Cambridge University Press.
- [15] Schupp, B, Smith, S, Wright, P, and Goossens, L. Integrating human factors in the design of safety critical systems - A barrier based approach. W. Johnson and P. Palanque. Human Error, Safety and Systems Development (HESSD 2004). Vol. 152, 285-300. 2004. Springer.
- [16] Schupp, B. A., Hale, A. R., Pasman, H. J., Lemkowitz, S. M. L., Goossens, L., and Fadier, E. Design support for the systematic integration of risk reduction into early chemical process design. Safety Science. 44, 1, 37-54. 2006.

- [17] Schupp, B. A, Lemkowitz, S. M. L, and Pasman, H. J. Application of the Hazard-Barrier-Target (HBT) Model for More Effective Design For Safety in a Computer-Based Technology Management Environment. CCPS ICW: Making Process Safety Pay: the business case. 287–316. 2001.
- [18] Svenson, O. The Accident Evolution and Barrier Function (Aeb) Model Applied to Incident Analysis in the Processing Industries. Risk Analysis . 11(3): 499-507 . 1991.

List of Figure Captions

Figure 1. Approach Diagram

Figure 2. A typical H-B-T diagram. PH1 is a primary hazard symbol, T1 a target symbol, and B1 & B2 are barrier symbols

Figure 3: SML representation of (a) fire hazard, (b) a human error hazard, and (c) recursion. FH1 & FH2 are functional hazard symbols, MB1 a mitigative barrier, PB2 a protective barrier

Figure 4. An ICO model of the operation of a pump and its motor (see section 3 for full case study)

Figure 5. Simplified Layout Diagram of Waste Fuel Delivery Plant

Figure 6. Fuel Hammer Effect

Figure 7: Piping and Instrumentation Diagram of fuel line. See text for explanation.

Figure 8. Petri net system model of the fuel line

Figure 9. SML representation of hazards, barriers and targets discussed in this text (PB1 and PB2 modelled in this paper)

Figure 10. PB1 Pipe Priming Procedure Barrier

Figure 11. PB2: Auto shut down of pumps after 3 minutes barrier

Figure 12 Steps 2 and 3 of the approach – barrier modelling and integration.

Figure 13. System model + PB1

E-voting: Dependability Requirements and Design for Dependability

J W Bryans

School of Computing Science,
University of Newcastle upon Tyne, UK
Jeremy.Bryans@newcastle.ac.uk

P Y A Ryan

School of Computing Science,
University of Newcastle upon Tyne, UK
Peter.Ryan@newcastle.ac.uk

B Littlewood

Centre for Software Reliability,
City University, London, UK
B.Littlewood@csr.city.ac.uk

L Strigini

Centre for Software Reliability,
City University, London, UK
L.Strigini@csr.city.ac.uk

Abstract

Elections are increasingly dependent on computers and telecommunication systems. Such “E-voting” schemes create socio-technical systems (combinations of technology and human organisations) that are complex and critical, as the future of nations depends on their proper operation. Thus heated debate surrounds their adoption and the possible methods for making them demonstrably dependable. We discuss the dependability requirements for such systems, and the design issues in ensuring their satisfaction, with reference to a recent proposal that uses cryptography for fault tolerance, in order to avoid some of the perceived dangers of electronic voting. Our treatment highlights the need for considering the whole socio-technical system, and for integrating security and fault tolerance viewpoints.

1 Introduction

Many electronic voting methods are currently being investigated in many countries. Brazil held its first fully electronic national election in 2002. State-level electronic elections have been held in the US. Across Europe many countries have also trialled electronic voting systems.

“E-voting” has both potential advantages and risks. These systems can make the casting of a vote more convenient and may therefore lead to improved turnout. Electronic recording and counting of votes could be faster, more accurate and less labour intensive. Digital technology could also provide greater anonymity than conventional approaches.

On the side of risks, the scenario of integrated computer and communication systems performing all functions from collecting the voter’s opinion (without paper records) through transmitting and counting them raises the possibil-

ity of large-scale vote-forging and/or spying on voters (and thus coercion or vote-buying). The possibility of these systems being implemented on current off-the-shelf computing platforms and with the low assurance standards common in much of the software industry has caused many experts to voice a concern that huge risks are being taken and spurred demands for voter verifiability of the functioning of voting machines, and guaranteed possibility of recounts, usually via paper records.

Any national or state election is a large, complex and important system, involving millions of voters and thousands of officials, as well as being increasingly reliant on IT. It is of immense importance that these systems be dependable. In this paper, we discuss the conventional dependability attributes of an e-voting system, including accuracy, secrecy, availability, and discuss the design concerns involved in guaranteeing them. Our focus is on proposed methods for reducing the current hazards in elections using voting machines in secret booths, not on the more extreme proposals for internet or mobile-phonated based voting. However, our discussion of requirements has a very general scope. We also consider the issue of reputation, and show that attacks on the reputation of a electronic voting system may potentially be more damaging than attacks on the system itself.

The paper is organised as follows. Section 2 discusses the balance between design time and run time measures for achieving and assessing the dependability of such a complex system. Section 3 describes the high-level dependability requirements for such a system. Section 4 introduces the Prêt à Voter scheme [5] for ensuring tamper-resistant, secret, paper-less transmission of ballots. Section 5 proceeds to show the fault-tolerant design implied by the proposed scheme and discuss the allocation of dependability requirements to subsystems, for defence against both accidental fault and malicious attacks. Sections 6 and 7 further discuss

some peculiarities of this socio-technical system: the composite, human and technical recovery mechanisms required for detected failures, and the risk of “loss of reputation”. Our conclusions follow in Section 8.

2 Assurance

In security there is a common tendency to look for provable correctness – impossibility of successful attacks – but the community’s attitude has been evolving towards a more substantial role for fault tolerance and for probabilistic assessment (as highlighted by the increasing visibility of security in recent dependability conferences). It is considered essential that mechanisms be provided to detect, contain and recover from failures. These mechanisms need to be robust in the face of malicious as well as accidental threats.

There are a number of ways in which assurance in a system’s correct behaviour with respect to a specification can be achieved. These can be thought of as lying along a spectrum with pure verification at one end and run time monitoring at the other, with testing lying somewhere in between. It is clear that we should use all of these techniques in combination to achieve increased levels of assurance.

In order to verify a system we assume some model of its behaviour and subject this to various forms of mathematical analysis to prove that it will satisfy certain (formally stated) requirements. This is fine up to a point but suffers from a number of deficiencies:

- Our analysis will only be as good as the models on which it is based. Unless we have succeeded in ensuring that our models are entirely faithful with respect to the properties of interest, proofs about the models will not necessarily carry over to the real system.
- It is difficult to ensure that the verified system will correspond exactly to the fielded system. Even supposing that our models start off as being faithful representations of the real system and its environment, systems and their environments evolve. This evolution, which could include degradation, patches etc, can invalidate the original analysis.

On the other hand, run time monitoring (error detection) also improves assurance. It directly improves dependability, because detecting unwanted states (“errors”) within the system can be made to trigger containment and recovery mechanisms; and it will deter some possible attacks¹. But it also suffers various drawbacks:

¹We will follow the convention of calling *error* an undesired state inside the system, *failure* an undesired output of the system to the external world, irrespective of whether they are caused accidentally or by malice [1]. Errors may or may not cause failures, and the goal of fault-tolerant design is to reduce the probability of their causing failures.

- Monitoring can be difficult: you need to know exactly what to monitor, monitoring has to be accurate and, in a hostile environment, robust against subversion.
- Good response and recovery strategies may be difficult to devise and execute.
- Detecting a violation at run time may be too late. For example, once a secret is out, it cannot be recovered.
- Inappropriate or compromised behaviour may not be easy to detect. Some properties, by their very nature, are not amenable to run time monitoring within the system that needs to satisfy them. Information flow properties (e.g. an attacker cannot read information transmitted along a wire) are a case in point. It is often the case that no monitorable event occurs when an information flow is violated.

We see these issues very clearly illustrated in the context of digital and electronic voting systems. Many of the proposed and even deployed electronic voting systems depend heavily on claims for verified and tested code, but fail to provide even the possibility of run time monitoring.

The Prêt à Voter Scheme [5], which we discuss in Section 4, is almost at the other extreme: for the accuracy property, virtually no reliance need be placed in the deployed components and all the assurance comes from close monitoring of the behaviour of the components. For the secrecy and availability requirements some verification will be necessary. It is therefore more amenable than many other voting systems to a dependability or fault tolerance style analysis.

3 Electronic Voting Dependability Requirements

There are a large number of types of election, common examples being “First Past The Post” and “Proportional Representation”. Although the requirements we consider here will apply to all elections, the type of election itself will introduce its own complexities. For concreteness, in this paper we will consider a very simple scenario. We will assume several candidates, with each voter selecting only one, and the winner being the candidate with the most votes.

We will refer to the dependability of an “election system”, by which we will mean the socio-technical system – people, as well as physical, computing and communication resources – performing all functions from collecting the vote through to ultimately producing the final tally.

We outline in this section a template of attributes and dependability requirements, that one would need to elicit in order to design and assess an election system. The details

to fit into the template (down to numerical values of parameters) should be informed by the special type of election used in that society, the preferences of the society using the system and the specific threats that elections face in that society; politicians required to state these requirements would obviously need the assistance of disciplines outside computing and engineering.

The main requirements of an election system belong to two categories: accuracy and ballot secrecy. Ballot information should be transmitted and counted correctly, and the link between voters and the votes they have cast should be secret. Neither property is fully guaranteed in current elections, so a reasonable requirement for any improved election system is for counting errors and violations of secrecy to be within acceptable limits. Dependability requirements will be about sufficiently low probabilities of these limits being exceeded, given the environment in which the system is deployed and, in particular, the threat profile, e.g., the technical ability, aggressiveness, and willingness to take risks of the potential attackers. We will discuss these acceptable limits and dependability requirements in the next subsections. The primary dependability requirements about accuracy and ballot secrecy imply secondary requirements: the system must be robust and resilient in the face of accidental and malicious threats. The balance between imperiousness to attack (or avoidance of accidental faults) and ability to survive them (or tolerance of accidental faults) is a matter of design trade-offs; but in practice a large amount of the latter seems necessary, since the system needs to survive highly competent and motivated attacks, and to be trusted by the general population. If the system does fail with respect to the primary requirements (i.e. accuracy or secrecy requirements are violated) in an election, it is essential that the failure be detected and flagged, so that it can be properly dealt with.

It is also essential for such a system to gain public trust in its accuracy and secrecy. One way to help engender confidence in the accuracy of the system is to provide voter verifiability: some way for the voter to assure themselves that their vote has been accurately included in the tally.

An important difficulty with these requirements is that naive implementations of verifiability would immediately violate secrecy, by requiring that votes be traceable back to the voters, or that a voter declare his/her vote when claiming that it has been miscounted. So, “end-to-end” checks, comparing the output of the system against its input, are not feasible without jeopardising secrecy. Assurance of accuracy must thus rest on assurance (by prior verification and/or run time monitoring) of the proper functioning of the mechanisms meant to protect it.

3.1 Accuracy

At the most abstract level, we would like the outcome of an election to accurately reflect the “intentions” of the eligible electorate. At this level we would need to consider social and psychological issues that might, for example, favour certain sectors of society, bias voter choices or encourage voter error.

In this paper we will restrict ourselves to the purely technical question of ensuring that votes counted in the final tally accurately reflect votes cast. We will assume that issues of authentication and prevention of double voting have been addressed.

Scientific studies of dependability, particularly of software-based systems, have long exhibited some tension between “perfection” and “good enough”. It has sometimes been said that a computer program can be made fault-free so that it will never fail during its life: from a dependability perspective it is “perfectly reliable”. Claims for complete perfection of this kind are now rarely, if ever, made. Instead, it is generally recognised that programs of even modest complexity may contain faults that will show themselves as failures at some time during the operational life of the system. Once this view is taken, questions about the acceptability for use of a system will involve dependability attributes such as reliability and safety: e.g. they will address questions such as “will it fail sufficiently infrequently?”

These considerations also apply to voting systems. Whilst we would *like* to have perfect accuracy, i.e. a complete *guarantee* that the result of an election reflects in all respects the voting intentions of the electorate, this seems an unrealistic goal in practice². Rather, we need to know that a result is *sufficiently* accurate, or (expressing it slightly differently) that sufficient confidence can be placed in the result.

In order to be able to talk about the “dependability” of a voting scheme we need to put some flesh on the bones of informal concepts like “sufficient”. The key here is uncertainty. There will be uncertainty about the nature and number of faults in the voting system, there will be uncertainty about the kinds and frequency of malicious threats it might meet. The result will be uncertainty in the relationship between the reported result of an election and the “true” result. As usual in dependability, the appropriate calculus for uncertainty is probability.

There are three potential sources of uncertainty: collection, transmission, and counting of votes. Even if we knew the intentions of all voters, there would be some uncertainty as to what is collected, and in how that is transmitted; even

²This remark is not intended to imply that approaches that seek perfection are invalid. On the contrary they may be plausible means of *achieving* dependability. For example, it may be possible to prove the absence of a particular class of faults: a kind of conditional perfection.

if we knew what is transmitted, there is some uncertainty as to how the outcome of the election is decided (for an example of this in paper-based elections, consider the variations in recorded totals between successive recounts in closely contested elections).

In any calculation, all the variables involved (votes as cast, votes as transmitted, etc.) would be *random* variables. So any measure of the discrepancies must be a stochastic one. For example, we might say that a voting procedure is “sufficiently accurate” so long as the *expected* proportion of votes reported for every candidate does not differ from the actual proportion by more than 1%. Alternatively, we might require that there is less than a 5% chance that any difference between a reported proportion and the actual one exceeds 1%.

Clearly, other formulations of what we mean (i.e. require) by *sufficiently* accurate are possible, even in this extremely simple example. In more complex voting schemes this issue of deciding what we mean by “sufficiency” may turn out to be quite difficult, but obviously necessary for a rational choice of system design.

Another difficulty in voting schemes is that the input for a particular election will generally be unknown: the confidentiality requirement will see to that. So, directly checking whether a particular election result is sufficiently accurate will be difficult or impossible. Of course, there are many systems for which we cannot know the exact value of the input data, but here we are specifically forbidden to find it out, by the requirement of confidentiality.

On the other hand, we can *test* a voting scheme to see how it behaves when presented with known inputs, and even with selected types of attacks or attackers (“tiger teams”). This will give some information – deterministic or probabilistic – about its dependability with respect to accuracy requirements, either conditional on the scenario adopted on testing, or unconditional, assuming a distribution of inputs and threats. Techniques akin to the various forms of software testing and fault injection have their place here.

3.2 Ballot Secrecy and Voter Anonymity

It will typically be a requirement that the way any individual voter voted remain secret. Besides the natural desire for privacy, ballot secrecy serves to prevent coercion or vote buying. The key point is that there should be no way a third party can determine which way a voter voted, even with voter cooperation.

Note that absolute assurances of total secrecy may not be realistic here. In certain exceptional circumstances secrecy will be violated: for example, if all the votes went one way.

Instead of ballot secrecy we might require voter anonymity. At first glance one might suppose that these are equivalent. We define anonymity to be that the observ-

able behaviour of the system remains the same under arbitrary permutations of the input ballots. This approach is formalised in [6] using the process algebra CSP. Using this definition, the scenario of everyone voting for the same candidate would still be deemed to satisfy anonymity but would fail the ballot secrecy requirement. Given that such a scenario is perfectly admissible and that the violation of ballot secrecy seems inevitable, this would seem to suggest that voter anonymity is the more appropriate requirement.

We will continue to use the informal name “secrecy” for the set of attributes and requirements about constraints on what one can learn about another’s vote.

How would one define dependability requirements with respect to the secrecy attribute? Vote secrecy is limited by many factors outside the election system proper, so that a practically zero probability of secrecy violations in the election system itself, even if possible, may not be required. Instead, we will have a notion of “secret enough”: a bound on the number and pattern of compromises of secrecy that are rare and limited enough not to endanger the general integrity of the process. However, this bound will depend more strongly than the one for “accurate enough” on the situation around an election: e.g., in a society where reprisals against opposition voters are likely and severe, the knowledge that even a small sample of votes can be known to the government may be enough to allow widespread intimidation.

3.3 Voter interface issues

An e-voting system that is used by a large and diverse group of people must have a readily understandable user interface, and the tallying and auditing mechanisms must be thoroughly understood by the people responsible.

More subtly, the voting procedure should not introduce any bias into the choice that the voter makes. If the ballot includes questions that are not of interest to a voter, then he or she may tend to choose (for example) the first of a number of options. An assertion about the lack of bias of a voting procedure (procedural invariance) would need to be substantiated by a cognitive argument.

3.4 Verifiability

A voter should have grounds to trust that his or her vote has been properly counted. Further, if it has not been properly counted, the voter should have a means of recourse to demonstrate this. However, this means of recourse must not be able to be used by a third party to coerce the voter into revealing his or her vote.

Demonstrating to yourself that your vote was counted is important, but it is also important that any voter can be assured that all votes are counted. This is called universal

verifiability in [7]. These properties go a long way towards providing transparency of the mechanisms, and their presence in a voting system would make an important contribution to an argument for public trust.

4 The Prêt à Voter Scheme

We now present an overview of the Prêt à Voter scheme. For full details see [5]. Prêt à Voter is based on the Chaum scheme [3], but uses a radically different mechanism to represent the encrypted vote value in the ballot receipt. In place of the visual cryptographic techniques of the Chaum original, the voters are provided with a familiar-looking ballot form. The voter makes her selection in the usual way by placing a cross against the candidate of choice. Thus a ballot with a vote for the Sophist candidate is indicated thus:

Nihilist	
Buddhist	
Anarchist	
Sophist	X
Solipsist	
	<i>7rJ94K</i>

To cast the vote, the voter now separates the right hand (RH) and left hand (LH) strips. The LH strip should be discarded, by, for example, feeding it into a shredder. The RH strip is placed under an optical reader or similar device. This records the information on the RH strip: the random-looking value at the bottom of the strip and the position of the X, i.e., the numerical representation of the cell into which it has been entered. The RH strip is now returned to the voter to retain as her receipt:

X
<i>7rJ94K</i>

The ballot forms would be augmented with various anti-counterfeiting devices, and a digital signature applied to the receipt when the vote is cast.

Thus far, aside from the retention of a receipt, the process of casting a vote is entirely familiar, to a UK voter at least. Now, an objection at this point is that possession of a receipt would open up the possibility of coercion or vote-buying. The trick that sidesteps this is that the order of candidate lists on the ballot forms are randomised. Choose a ballot form at random, and the order in which the candidates are shown will be unpredictable. Clearly, with the LH

strip removed, the RH strip alone does not indicate which way the vote was cast.

Now the problem is how the votes will be extracted and counted. This is where the random value printed on the bottom of the receipt comes into play. Buried cryptographically in this value is the information needed to reconstruct the candidate list shown on the LH strip and visible to the voter when they cast their vote. This information is encrypted under the secret keys of a number of *tellers*. These tellers are automated, but have a similar role to the human tellers that count the votes in a manual election. Thus, only the tellers acting in consort (in an anonymising mix [4]) are able to reconstruct the candidate order and so interpret the vote value encoded on the receipt.

Once the election has closed, all the receipts are transmitted to a central tabulation server which posts them to a secure web bulletin board (WBB). This is a write-only, publicly visible facility. Only the tabulation server can write to this and, once written, anything posted to it will remain unchanged. Voters can visit this WBB and confirm that their receipt appears correctly.

After a suitable period in which voters can verify that their receipts have been correctly posted, the set of tellers take over and perform a robust, anonymising, decryption mix on the batch of posted receipts. Intermediate stages are also posted to the WBB for partial random audits, and so is the final list of decrypted, anonymised ballots.

In summary, the *tellers* perform a set of publicly available algorithms, aimed at guaranteeing that all votes are properly decrypted and also that accidental or malicious vote tampering (altering or deleting ballot contents or adding spurious ballots) has only a minuscule probability of going undetected. We omit the details of this here, but they can be found in [5].

All this is fine as long as all the steps are performed faithfully. If we are prepared to trust the entities executing this process then we can be confident that the election will be accurate and the vote values kept secret. However, the aim of schemes like the ones devised by Chaum and Neff and Prêt à Voter is to achieve these goals without the need to place such trust in any of the components of the scheme. In section 6 we outline the mechanisms used to detect any malfunction or misbehaviour by the devices or processes that comprise the scheme.

5 Satisfying the dependability requirements

The function of the election system is to transmit the information from the voting booth to the counting stage. Prêt à Voter (like Chaum's scheme [3]) is remarkable for its *error detection* mechanisms, which, when coupled with appropriate error treatment, allows a highly dependable system to be built out of undependable components. Assurance in

the error detection mechanisms themselves, in turn, requires proof of properties of their algorithms, plus verification of their implementation.

Accuracy is assured by a series of checks on the mechanisms that perform stages of the transmission and counting. Some are performed directly by the voter in the booth (Section 6.1). The posting on websites of voters' (encrypted) retained receipts, together with people *actually checking* the posted data against those left with the voters detects errors during the transmission of the votes from the booth to the initial teller. The auditor detects accuracy errors in each decryption-based stage of the transmission.

So far, we have not considered denial-of-service attacks. In a viable design, detected errors in the core system (that part of the system defined thus far) must lead to attempts to recovery, repeated if necessary. Eventually, therefore, either the result is considered valid by the error detection mechanisms (and can thus be either a real success or a secrecy failure or an accuracy failure) or the election does not complete.

To evaluate the whole election system, it is necessary to consider the threat profile: probability of attacks of each kind. Importantly, this is affected by social and cultural factors and by the would-be *attackers'* perception of the effectiveness of the error detection and/or recovery mechanisms, and of the severity of consequences that they can trigger for the attackers. This assessment requires understanding of social and psychological factors of deterrence that are certainly outside the competence of reliability engineering. Yet, this dependability analysis suggest clearly which questions the decision makers need to ask their social scientists, historians and psychologists.

In the high-level system design stage, a designer uses the dependability requirements (Section 3) to apportion responsibility to the various subsystems, and judge whether the subsystems proposed are suitable for the system to satisfy the overall system requirements. For accuracy and secrecy failures, $Pr(\text{undetected failure}) \leq Pr(\text{error in core system}) \times (1 - \text{coverage})$ where the *coverage* of the error detection mechanisms is their probability of detecting an error, if an error did occur. Given the overall requirement (left-hand term), high-level design must determine the two right-hand terms. This will typically be done separately for different categories of failure modes and causes. Errors may be caused by misfortune (accidental faults) or malice (attacks, also called intentional faults). For the former, familiar techniques can be applied towards a conservative assessment of their likelihood and the coverage of the error detection mechanisms. The threat profile is an input for the designer of the detection systems that have to assure the required coverage. Technical analysis of Prêt à Voter gives a high coverage, conditional on assumptions on the type of attack (e.g., no collusion between

multiple tellers and/or auditors) that must be satisfied via further technical and organisational mechanisms.

As for non-completion failures (*ncf*), if the reaction to a detected error were to abort an election we could write $Pr(ncf) = Pr(error)(coverage) + Pr(false\ alarm)$

We observe that shifting responsibility for avoiding undetected failures from the security of the mechanism collecting the votes to high detection coverage may shift the focus of some adversaries to causing non-completion failures, by increasing their attacks on either the core system or the error detection mechanisms. On the other hand, for a given strength of the security of the core system, adding detection mechanisms will have the desired result of reducing undetected failures and increasing the risk for attackers, but will still increase the probability of non-completion failures, via possible errors in detection mechanisms.

6 Recovery Mechanisms and Strategies

The core scheme provides a high level of assurance that any accidental or malicious corruption of votes will be detected. But these error detection mechanisms by themselves do not reduce the probability of failures. Detected errors must also be dealt with properly; aborted elections are still failures.

In this section we propose some possible strategies and explore their implications for the robustness of the scheme.

Leaving aside for the moment problems that might arise outside the core system, there are basically three failure modes with respect to the accuracy requirement:

- Ballot forms might be incorrectly constructed in that the information buried in the cryptographic value on the receipt might not in fact match the candidate order shown on the LH strip.
- The receipt might be incorrectly recorded or transmitted to the WBB.
- The tellers might fail to correctly decrypt the receipts.

For all of these there are random auditing mechanisms in place to (probabilistically) detect any malfunctions or corruption. Details can be found in [5].

Turning to the effects of attacks, our response strategy should be based on patterns of detected errors rather than isolated errors. It is possible for individual errors to look quite innocent and yet a group of errors taken together may constitute a collusion attack resulting in the deliberate corruption of a vote. In defining our response strategy therefore, we must take account of the nature and patterns of errors. This is reminiscent of the challenge faced by intrusion detection in general.

6.1 Ballot form errors

Voters are urged to confirm that their ballot sheets appear correctly in the input column of the web site. If the voter fails to find a copy of their receipt posted or finds that the posted receipt does not match their copy, then they should report this. The process of collecting and collating such reports needs to be handled carefully: the dependability of the scheme depends on appropriate response to such reports. It is important that voters are made aware of a recognised and dependable reporting authority, for example the returning officer.

It may be necessary to check voter claims of missing or corrupted receipts. This can be done by requesting voters to provide their receipts and confirming each report. If a particular booth is confirmed to have lost or corrupted a number of ballot receipts, it should be assumed to be either defective or malicious and taken out of service. It may be necessary to rerun votes cast at that booth.

6.2 Teller errors

The auditor performs checks on a random sample of links on the WBB for each teller to establish whether or not the decryptions have been correctly performed. All resulting errors need to be collated and analysed.

Requirements can be set on the tellers to deploy reasonable quality computing systems and cheap fault-tolerance to achieve very low probability of accidental corruption of ballots. The purpose here is not to avoid undetected errors at the system level, but to make even minor disruption of elections by accidental faults very unlikely, and thus: protect society's trust in the election system; prevent non-completion failures; and ensure that only massive effort by attackers can disrupt an election, and that such attacks cannot be disguised as technical "glitches". This improves diagnosis (discriminating malicious from accidental faults) and thus also improves society's confidence.

6.3 Secrecy failures

The notion of a detector for secrecy errors raises interesting issues. The main question is: what would such a detector detect? There are clearly plenty of blatant errors that could be detected. If say a teller applied the null permutation then this would be picked up at audit. On the other hand there appear to be a large class of secrecy failures that would not manifest themselves in the target system, especially when we note that there are restrictions on where we place our monitors.

Consider the following: we have just two tellers and they are in collusion: the first applies a "random" permutation

and the second applies the inverse. This is clearly a violation of secrecy, or at least puts the system in a hazardous state w.r.t. secrecy.

It could be argued that if nobody but the tellers knows that this has occurred then no failure of secrecy has occurred. There is always the danger that this will leak out at some time to some group later. Also, the fact that such a fault could occur with our having no way to detect it is itself worrying and has the potential to undermine confidence. The fact that we may not be able to detect such hazardous states also means that it is hard to demonstrate that such a state has not occurred. Absence of an alarm does not imply absence of error state.

We should note that we could in principle audit all links and detect this attack, but this would then immediately violate the secrecy property we are trying to preserve.

7 Trust and Reputation

One of the primary assets of an election system is its reputation. Indeed, the reputation of the election system is an important factor in the mandate of any person elected by it. An attack on the reputation of an election system could be as damaging as an attack on the system itself.

One way of thinking about the reputation of an election system is in terms of the confidence that the voters place in it. We can be more precise, and say that voters have requirements of election systems expressed as dependability claims ("its accuracy is no worse than ... , its secrecy is no worse than ..."). One can never be sure that such claims are true, but on the basis of the available evidence one will have a certain *confidence* in their truth. We can expect that if confidence were to fall below a certain threshold, there could be serious social implications.

These ideas are close to recent formal approaches to confidence in reliability and safety claims for systems [2]. There, an attempt is made to model confidence as a (possibly subjective) probability that a dependability claim is true, based upon the evidence and reasoning of a dependability "case". The novelty in a voting system is that *intentional faults*, as well as accidental ones, can contribute to a reduction of confidence in a dependability claim.

Just as worrying as the scenario of public opinion mistrusting a dependable system is the scenario of a populace trusting an undependable system, leading to elections being stolen through vote tampering or vote buying. Efforts to avoid the first danger might well increase the likelihood of second. An adversary could even pursue strategies like minor sabotage of a few elections, causing many minor error reports which governments will be compelled to play down, so reducing the public's sensitivity to reports of problems, and reducing the probability of appropriate reactions to the real, all-out attack when eventually launched.

A comprehensive threat profile should consider attacks against reputation. This highlights again the importance of integrating the considerations from the social and human sciences, that should inform the dependability requirements for the election system, with the technical considerations from reliability and security engineering (including human factors and organisational issues) that determine the feasibility of satisfying the requirements via the various possible designs of election systems.

As an example of a possible attack against the reputation of the encrypted ballot receipt, consider the case where an attacker wishes to coerce people into voting in a particular way. He could (falsely) claim that he is able to decrypt the partial receipts retained by the voters, and insist that they are surrendered to him. Voters would then have to choose between believing the experts that told them the receipt was undecipherable, or believing the coercer.

Defences against such an attack might include things like specific public information measures to build public trust. These defences would necessarily need to be psychologically and sociologically informed, and indeed building a comprehensive threat profile would also require interdisciplinary thinking.

The reputation of a voting system could also be damaged without a malicious “attack”, e.g. by even minor accidental errors being detected and requiring the telling procedure to be rolled back: the transparency of a system may work against the system on a societal level.

Attacks on the reputation of a system are not new, nor is defending against them. What makes the reputation of an electronic voting system interesting is the criticality of conveying the supporting arguments to such a large number of people. Compelling dependability arguments are usually designed to convince experts; here the general public must be convinced.

8 Conclusion

We have discussed at a high level the dependability requirements of an election system, which includes computer and communication technology as well as the people and organisations controlling it.

With its large scale and tight interconnection of technical and human components, an election system exemplifies the kind of systems that will become increasingly common. It is comparable in scale to some telecommunication networks or corporate IT systems, but is more similar to an anti-missile defence system in being an on-demand system with very stringent requirements on the single demand. It differs from all these other examples in having limits on its error detection capabilities imposed by a secrecy requirement, rather than just by physical constraints or costs; and its usefulness depends more heavily than with many other

systems on preserving its reputation with the public, which is vulnerable to many kinds of attack.

The solution we have used as reference in our discussion relies on a combination of provable properties of cryptographic algorithms, and extensive error detection methods which in turn depend on both technical and social mechanisms for their functioning.

Our discussion has emphasised three aspects: requirement specification, with the need to translate society’s informal requirements and to consider how threat profiles change compared to those affecting non-electronic elections; design issues for the fault-tolerant system, including the need to complement the basic cryptography-based ideas with explicit methods for assurance of the detection mechanisms and explicit recovery mechanisms, subsystem-level dependability requirements and the concern for denial-of-service attacks; and the need to integrate technical considerations with psychological and social ones that determine the threat profile, the voters’ reactions and the effectiveness of the socio-technical mechanisms for error detection and recovery.

Acknowledgments This work was partly funded by the UK Engineering and Physical Sciences Research Council (EPSRC) through DIRC, the Dependability Interdisciplinary Research Collaboration. We wish to thank our DIRC colleagues at Newcastle and City Universities for many stimulating discussions on this topic.

References

- [1] B. R. A. Avizienis, J.-C. Laprie and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable And Secure Computing*, 1(1):11–33, 2004.
- [2] R. E. Bloomfield and B. Littlewood. Multi-legged arguments: the impact of diversity upon confidence in dependability arguments. In *International Conference on Dependable Systems and Networks (DSN)*, 2003.
- [3] D. Chaum. Secret-Ballot Receipts and Transparent Integrity: Better and less-costly electronic voting at polling places. <http://www.vreceipt.com/article.pdf>.
- [4] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–88, Feb 1981.
- [5] D. Chaum, P. Ryan, and S. Schneider. A practical, voter-verifiable election scheme. Technical Report CS-TR-880, University of Newcastle upon Tyne, 2004. <http://www.cs.ncl.ac.uk/research/pubs/trs/papers/880.pdf>.
- [6] S. Schneider and A. Sidiropoulos. CSP and Anonymity. In *ESORICS*, volume 1146 of *LNCS*, 1996.
- [7] B. Schönmakers. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Advances in Cryptology - CRYPTO’99*, volume 1666 of *LNCS*, pages 148–164, 1999.

Time as a dimension in the design and analysis of interactive systems

M. D. Harrison¹ and K. Loer²

¹ Informatics Research Institute, University of Newcastle upon Tyne, NE1 7RU, UK
Michael.Harrison@ncl.ac.uk

² Department Strategic Development, Germanischer Lloyd AG - Head Office, Voserstr. 35,
20459 Hamburg, Germany
loe@gl-group.com

Abstract. This chapter discusses the relevance of timing to the design of interactive systems. It introduces a set of dimensions that can be used by designers to assist the process of thinking about interactive systems. It explores examples of analysis of interactive properties of a system design using a specification based on uppaal and in particular assesses design choices that arise from timing consequences of a design.

1 Introduction

Timing is an important and yet neglected feature of the design and implementation of interactive systems and in understanding their usability. People use technology to complete activities, to achieve goals, to judge whether to delay goals, and to do things imperfectly under time constraint. People's experience within an environment involving computer based systems may be affected by temporal concerns. For example, in the context of an airport information system, travellers may become anxious if the status of a flight is not updated often enough, or may become annoyed if they are reminded too often. Time requirements may relate to experience and may be subjective or requirements may be critical to the safety or security of a system and therefore objective. This chapter highlights the importance of time in design and suggests ways that devices within systems may be designed to support aspects of time in the user's interaction more effectively. Finally the chapter briefly illustrates the role that modelling techniques might play in helping designers to explore the timing implications of a design.

In practice time is an element of the *context* in which a device is placed that may affect its activity in a number of ways. It may be a factor to be taken into account in relation to the behaviour of the device itself. Properly designed systems may help people juggle activities to use time most effectively, may enable them to predict episodes that are likely to involve high workload and deal with them accordingly, may keep the user in control while taking away those aspects of the activity that can be dependably automated. Time is a resource to be used effectively. Systems may use timely feedback, or timely recommendations to allay the stress of not being sure whether some future event will happen at the appropriate moment.

The chapter has two objectives. The first is to discuss the way design may be affected by time and to explore dimensions that may help designers to account for time in

design. Therefore Section 2 discusses a number of time dimensions and Section 3 considers ways of thinking about how the system to be designed resources the user. Finally Section 4 explores the role that modelling may play in helping designers to identify timing issues in the design of a particular system.

2 Time dimensions for design

Design of the whole interactive system may be improved by a proper consideration of time. There are a number of dimensions that may be important in a consideration of how time might impact a design. These dimensions may be more or less important depending on time granularities, the nature of the context in which the device is situated, the training and capabilities of the user and may be categorised as: internal / external; subjective / objective; sporadic / continuous; user initiative / system initiative; extreme / normal.

Internal / external The pace or timing of an interaction may be driven by the activity of the device itself. As a result user actions may take time and affect the pace at which the user can achieve goals. Alternatively the environment of the device may drive the pace of the interaction. For example, incoming email messages or telephone calls can affect work-rate. These characteristics of the system may be important to how the task is carried out and therefore device design may need to take this into account. For example, the device might alert the user to the consequences of the external environment or to help the user decide how an objective should be achieved given some external circumstance.

Subjective / objective Timing affects the *experience* the user has of the system by generating a sense of satisfaction or well being or by causing anxiety or a sense of hurry. Such experiences might be created without otherwise affecting the behaviour of the system. Such experiences are subjective. Those aspects of the system for which timing is externally important — for example, the system times out if the user fails to achieve an objective (entering a pin number) within a certain interval — may be said to be objective.

Sporadic / continuous Timing characteristics may be continuous. They affect the ongoing pace of the interaction. Alternatively effects might be sporadic — they happen in bursts where users have deadlines to meet with differing degrees of warning, and as a result may have periods of high workload. For example, the broadband connection may suffer high loadings that might be exhibited as a temporary cessation to the user's progress.

user initiative / system initiative The pace of interaction might be driven through the initiative of the user or through the activity of the device.

Extreme / normal Timing effects may be normal and dealt with in the everyday activity of the user or something that only happens in extreme situations. In extreme situations unusual decisive action may need to be taken leading to changes in the pace of the interaction. These changes may be hard to deal with and may require new strategies or plans that are unfamiliar, and the device interface might provide assistance to the user.

Thinking about these design dimensions can help to understand how the system should be designed, and they may lead to a rationale for design decision.

Internal versus external design An internal / external timing dimension might lead to a number of design decisions. For example:

- Designing a text reader so that the first few pages are displayed to mask the delay involved in the computer reading the whole document. This is a design decision that is made in response to an internal timing effect. How necessary or effective such a strategy is depends on the activities of the user. If the user normally starts reading at the beginning of the document then this might be a good strategy leading users to a perception that there has been no delay. Hence an appropriate design strategy involves an understanding of delays in the system, generation of information and the way that the user operates.
- Providing a public display in an airport terminal that continually updates to show when a flight is to leave or is guaranteed to provide flight information with some specific delay after arriving in the hall. Here timing considerations in the external environment are important to the tasks being carried out by the user of the display. The currency of this information is important to the effective performance of the user's (passenger's) task. A similar but more direct purpose is achieved by a pedestrian traffic light that counts down to indicate how long before the light goes green. Both examples provide information about how much "slack time" is available in which the user can carry out other activity.

Subjective versus objective design The subjective / objective dimension is more difficult to assess. Consider again the airline display. An objective requirement of this system would be that the information is updated within some maximum time interval, whereas a subjective requirement would be that the display should be updated sufficiently regularly for the particular passenger so that the anxious traveller may be sure that the information presented on the display is a sufficiently accurate picture of the system.

Sporadic versus continuous design Here the design might enable items arriving sporadically to be buffered so that the user can deal with them in a more continuous way, or enable the user to batch items so that when the system is able to deal with the actions they can be carried out. Another example is a control system such as an aircraft where automation automatically takes over from the pilot when the risk of the automation performing the actions are less than the pilot carrying out the actions — for example high performance manoeuvres in which an aircraft is inherently unstable.

User initiative versus system initiative A system might be designed to enable the user to see what progress is being made toward the goal so that even though the user has the initiative, the system will provide information about the consequences of the user's pace. An example of such a system is a car trip computer that predicts time to

destination at current pace. Alternatively the system may calculate that, according to current progress, the user will not meet a deadline and therefore carry out some of the processing even though the quality of the output may be more prone to error as a result.

Extreme versus normal design Here an infrequently practiced procedure might be supported more directly by the interface than a routine procedure. Consider the support for standard operating procedures for normal operation in an aircraft cockpit versus recovery procedures that are rarely carried out.

Good time support can be an important contributor to improving user-system interaction. The design may be improved by providing resources for the user to enable them to manage aspects of time. Conn [1] comments that time design should be considered in terms of the tasks to be carried out. Time design is however also about experience, such as reassurance that appropriate resources will be provided in a timely manner so that the system provides an acceptable experience. From a task perspective timing may have an impact that a design should take note of, for example:

delays the design may help the user to manage the delay by indicating its extent, by counting down, by visualising how much of the delay has been spent. As well as providing information about the delay itself, the device can help the user manage available slack time. For example an airport based information system might inform a passenger of the existence of a restaurant if a flight delay has occurred and there is sufficient time to have a meal.

deadlines in an interaction. It may not be enough to say when the deadline is but be more appropriate to help people manage the deadlines. Hence it might be appropriate to:

- give a prediction of whether the objective will be achieved at the present rate of progress;
- indicate how pace may need to be changed in order to achieve the deadline;
- offer to carry some of the workload to achieve it faster;
- indicate what kind of result will be achieved if current progress is continued.

pace of the interaction where the device may take over some of this activity if the pace is too slow.

deadlines the system may indicate explicitly to the user how to defer the inessential so that the essential can be carried out before the deadlines. Alternatively it may indicate how good the result is as the system continues to refine the result as the deadline approaches.

From the external perspective design depends on the effects of external phenomena. Good design based on time will depend on:

- Activity arrival rates and how predictable these are and the deadlines associated with carrying them out
- The user's or users' awareness of activity arrival, their control mode (see for example [2]) which gives an indication of how much time there is to reason about options and their awareness of activities
- How the activities that are external relate to system objectives, what resources are available and what their service rates are

- The pre-emptability of activities that have to be performed, whether it is feasible to combine activities, interleave them, postpone or drop them. The discretion that there is for satisficing and trading off between system objectives.

Systems and devices can be designed to make these issues more explicit to the operator.

3 Resourcing time

People perform actions by using resources that are available to them in their environment. These resources might be information about the actions that are possible, information about goals, about plans, about the current state or about the effects that action might have [3]. Time design may be facilitated by identifying and analysing resources that have an impact on the temporal behaviour of the system. These resources play a role in shaping the interaction. The timing and availability of these resources may be a critical factor in the user's pace and understanding of deadlines and delays. In the following list, the impact of time is considered from a resource perspective.

- plans specifying actions to be performed and the order in which they are to be performed. It may be necessary to know whether the order of actions can be changed, whether it might be possible to drop actions and what the effect on the overall goal would be of dropping the action, whether there are alternative strategies that will produce more or less reliable answers depending on time pressure. While all these timing properties all relate to performance against goal it may be reassuring to know about the progress that is being made in relation to the achievement of the goal.
- goals and sub-goals to be achieved: the effects of the states of the system that are to be attained. In terms of goals it may be appropriate to know whether a sub-goal is essential to the achievement of the top-level goal so that a decision can be made to drop it if appropriate. It may be helpful to know how long it will take to recover a sub-goal.
- knowledge of the current state of the world or interactive system to be used as a means of comparison with the goal state that is to be achieved The current state may in fact differ from the state as represented by the device configuration because updates have not been sufficiently recent. As a result it may be difficult to anticipate what still has to be done and how long it will take to do it.
- historical information about process, actions and properties that have held of the state in the past; this may be in the form of a script of the last few commands as is the case of Unix or some description of previous landmarks or profile of how the current state was reached
- the effect that actions may have on the system or the world and how long these actions take, and how long they take to undo
- action possibilities that the system currently supports (including constraints on the interaction that limit what can or cannot be done) including indications of best strategy relating to the possibilities in the face of temporal concerns.

Resources may be used to control the plans of the user and to support the best strategy under time constraint. Resources may be used to indicate the immediacy of a deadline and the impact that actions may or may not have in achieving the deadline with an appropriate level of accuracy. Hence a plan following interface where the device is forcing the user down a path can take control away from the user and may affect the mode of control through the pace of the interaction. The operator may no longer have time to consider the activity nor the freedom to choose alternative courses of action.

Conn's *affordances* [1] are intended to help designers to provide "timely" information in the context of task performance: to support events such as the acceptance or scheduling of tasks, their initiation and completion, and making clear to the user when exceptions occur. Conn's recommendations also relate to status information about the task's scope (how big is it), progress, how much of the execution is left. In relation to events and status information, Conn discusses the role that a "time tolerance window" plays in enabling an operator to assess for example the length of time an operator allows before deciding that a task is not making progress or that something must have gone wrong. Implicitly the information about tasks, status and events leads to support for decision making. To complete things in time appropriate human decision making may be required. The ability to predict how long something will take, to alert people to the likelihood of delay, to compromise or make realistic trade-offs, in order to be timely is critical to effective and satisfactory work practice. When activity supported by the device is task related the appropriateness of different techniques for visualising and presenting this information depends on the kind of task that is being supported. For these sorts of systems to understand and design interactive systems it is necessary to design to help people in the system achieve their goal according to contextual constraints. This might for example involve understanding how scheduling takes place in order to help users to decide how the technical system should support this activity, what aspects of the activity should be automated, and how that automation should be perceived by the user, and how the system should support the decision making that is crucial to effective activity. Interactive systems are hybrid systems the computer system is embedded in an environment involving people as well as systems that are effectively continuous and which must be understood in the discrete terms of the computer system.

4 Modelling time for design

4.1 Introduction

Modelling aims to allow designers to assess the implications of a particular set of design decisions. The concern in this chapter is that models should be used to assess ways in which users of the system can make better use of timing aspects of the system. The chapter has so far described and illustrated features that may be important from a modelling perspective: how time relates to the control of the system; how information resources relate to temporal properties of the system; how the system affords delay, pace, deadline etc. These features can be designed in a variety of ways and for a variety of purposes and different kinds of model are most appropriate for the modelling of different features of the design. In this chapter a particular modelling technique is used. Other more quantitative styles of modelling would be appropriate for assessing typical

queueing behaviour for example, the identification of appropriate strategies for dealing with steady state behaviour.

To illustrate possible techniques, two examples are explored in more detail. The first is a system that involves operator control of a dynamic process (namely a paint shop). This activity is taking place at a time granularity that involves seconds rather than minutes. The system offers the user opportunity to automate some of the activities that have to be carried out. The system can break down from time to time, the paint guns wear out and require replacement. Options are provided for the operator to replace or repair parts. These two options have different costs associated with them, replacement is immediate but costs money while repair is free but costs time. Modelling is used here to explore what would be the optimal strategy in normal circumstances, what would be most robust to variation and what alternative strategies would be appropriate in different extreme circumstances. This analysis can be used to assess whether the interface to the operator is appropriate. The second analysis is concerned with the deployment of service information within a built environment, in this example an airport. The important activity from the user perspective works a time granularity that is in terms of minutes rather than seconds. The system uses public display and hand-held device. Here issues associated with the timeliness and freshness of deployed information are important. An example property that may be checked of the model is that relevant flight information should be available to a passenger within one minute of arriving in a new space.

Using modelling techniques, processes can be specified to describe the physical characteristics of the environment, assumptions about the user and features of the interactions with the device that is being designed. These processes can, in particular, be used to capture temporal characteristics of the external environment, strategies including temporal strategies relating to the user's behaviour, features of the artefact that are important from the perspective of interaction with it.

The uppaal tool [10] is used here. It allows the analysis of networks of linear hybrid automata with clocks whose rates may vary within a certain interval. This makes it possible to take the different temporal reference systems into account that may be present in the case study – for instance, the real-world frequency of items on the belt and the operator's perception of the frequency under varying workload. Automata may communicate either by means of integer variables (which are global) or by using binary communication channels. Messages can be passed and synchronised correctly employing modelling patterns channels to synchronise and communicate. These are described in the uppaal tutorial [?]. Communication occurs as a result of two process synchronisations using receiving actions $a?$ and sending actions $a!$. Guards are used to describe the circumstances in which communications can take place. Automata may be guarded by conditions involving clocks that can be used to represent delays or time invariants. It is not within the scope of this chapter to describe the syntax and semantics of uppaal in detail, however the examples given below should be sufficiently clear to give the spirit of the approach. Although the expressive power of the notation has some limitations, the system has the advantage of easy availability using a graphical interface and this makes the model more accessible to non specialists. Uppaal provides tools for the simulation of systems — the state transition diagrams are animated, and the inter-process communication is displayed as an animated message sequence chart. The tool also sup-

ports analysis by state exploration. Models used in model checking can only label input and output resources. A key issue in developing these models is to minimise the number of states by making appropriate abstractions so that analysis can be performed.

Simulation or checking of the model generates state sequences. They can be generated through a simulation facility. Through user intervention sequences can be created and explored that are the bare bones of scenarios that have been gathered through a process of user elicitation. These scenarios might have been gathered by interviewing users of the system that is to be replaced or as a result of some kind of experimental evaluation of how strategies emerge or of typical strategies. Alternatively sequences might be generated by asserting properties that can be checked by the model checker. By this means it could be possible to discover the path that would take the least time or the strategy that would minimise loss given failure of a particular component of the system — here loss might result from delay in carrying out some action. Once paths are generated further analysis can be carried out.

This technique is discussed in more detail in Campos and Harrison [5] and Loer and Harrison [6]. It is intended that domain experts or human factors experts consider the implications of a sequence generated through the modelling and checking process. They use their expertise to envisage a situation, a context, in which the sequence might occur and creating a narrative based on the sequence. Loer and Harrison [6] explore property templates or patterns of properties to make the process of generating appropriate properties more intuitive. Their tool provides an interface that allows the instantiation of CTL properties to a model based on usability heuristics [8] in order to make the process of analysis easier for designers. Reachability is of particular interest in the context of timing properties, can significant states (goal states) be reached within a timescale subject to given constraints. In the finite state model of the environment, end states can be associated with reaching significant points in the process, for example if a physical model of the process is involved then relate a state to the completion of that process.

Alternative paths may be explored by adding additional constraints to the property to be checked. Further analysis might involve the manual annotation of action sequences with the information resources that have been described as part of the design but can only be hinted at in the state labelling captured in the model (see [4]).

A key to effective modelling is to find appropriate abstractions without biasing the analysis. Device models at different levels of abstraction capture key characteristics of the interactive system. For example a flight management system [5] is explored in the context of a simple model of airspace trivially capturing notions of ascent and descent. This model of context is sufficiently general to explore mode issues associated with the design of the device. On the other hand, in [7], two context models constrain the behaviour of a handheld device. Here a sewage plant model characterises the behaviour of tanks, pipes, valves and effluent contained in the tanks and transported by the pipes. An additional model captures the spatial position of the hand-held device.

A human factors expert can consider alternatives to assess the implications for the design of the system and, possibly as a result, work with the designer to produce an interface that has better timing characteristics.

4.2 The paintshop study

The paintshop study is an example of a system that involves the control of dynamic processes. Indeed it has been used as a “microworld” experiment to explore how users devise strategies in the face of real-time constraint [9]. The design issue here would be to explore what device interface would provide most support for the user in the face of varying levels of workload as well as to support the decision processes required when choosing between repair and replacement strategies in the face of paintstation failure.

Hence modelling is used to explore alternative strategies. These strategies might include decisions in relation to postponement, interleaving, synchronisation, speeding up or slowing down of function servicing and whether control should be automatic or manual.

A number of questions then relate to the design of the artefact — how an appropriate strategy can be conveyed in the interface, whether appropriate strategies or next actions can be adequately resourced [3].

The system involves a conveyer belt that transports boxes to two parallel paint stations (Figure 1) for painting. Boxes may enter the paint system at different rates and a financial reward is given according to the number of boxes painted. When boxes arrive at a distribution lift, the user can then choose for allocation to paint stations to be automatic or to press the “up” and “down” manual buttons as appropriate. The painting process can either be set to automatic mode (which is the default) or to the manual mode. In automatic mode, the paint station will automatically specify the number of coats to be painted, carry out painting and wait for it to dry. The rate of paint flowing through the nozzles is displayed just above each production line. The flow rate may decrease if nozzles become blocked or increase if the nozzle leaks thereby providing information about the future potential for replacement or repair. To paint an item manually, the operator has to click on a box and keep the mouse button pressed for a specified period of time to complete the process. If the mouse button is released before the minimum paint time the box is not painted and a spoiled box is released.

In the model, painting takes five time units in the automatic case and two time units in the manual case. When a nozzle ceases to function properly it can be repaired or replaced. Replacing a nozzle incurs no time cost but does incur a certain monetary cost. Repairing the nozzle incurs no monetary cost but causes a delay before the nozzle can be used again. In the micro-world experiments the cost and time variables were manipulated and indeed in the model also these values can be manipulated. Depending on the rate at which boxes arrive at the station and the state of the nozzles and the strategy used to employ the paint stations a certain proportion of the possible boxes will be painted. Boxes can fail to be painted either because the appropriate procedure has not been carried out inside the paint station or because the queue of boxes waiting to be painted exceeds a certain number. When the queue waiting to be processed exceeds some number boxes are lost down a reject chute.

4.3 Modelling paintshop

The paintshop is modelled as seven concurrent processes with the aim of identifying what the optimal strategies under different constraints are and how the design may be



Fig. 1. The paintshop system

changed to make it easier for the operator to adopt a better strategy. Thus it is possible to express and check for reachability properties such as:

1. “Is it possible to reach a state where the clock x is greater than 20?”
2. “Is it possible to reach a state where all boxes have been painted?”

The result of checking the property is a path that can then be explored in more detail.

Two of the models (Figures 2 and 3) describe the physical environment of the system. A dispatcher automaton (Figure 2) captures the regular distribution of boxes defined by a constant (`workload`) that is used to describe the workload level. This can be changed to explore different workloads. This process dispatches objects to the incoming queue. Figure 3 specifies the behaviour of a box being channelled through the two paint

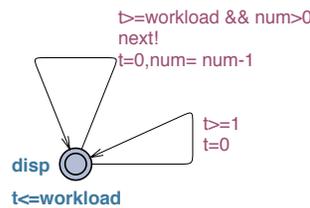


Fig. 2. High workload incoming belt

stations. This automaton models the part of the system containing the queue of boxes waiting to be serviced by the paint stations, as well as the lift that causes the boxes to be moved to one paint station or the other. It also models a repository for boxes that have fallen off the end of the queue and therefore lost. The final physical element (not illustrated) models the belt of finished items. When `workload=2` a new box arrives on the belt every two units (which is high workload). Values representing a medium and low workload are 3 and 4 respectively. In order to reduce the complexity of the analysis, the number of boxes in the model is limited to 10. While it is acknowledged that this is a great simplification in comparison to the continuous flows a real-world operator is likely to have to deal with, for the purpose of this analysis the simplified model is sufficient.

The device design is captured by the process in Figure 4. This describes the behaviour of the button that can be used to change from manual to automatic delivery to paint stations, as well as the feature that enables automatic or manual paint delivery and the mechanisms for repairing or replacing. There are two instances (`station1` and `station2`) of this process that describe the behaviours of the two paint stations. The description of automatic and manual operation is contained in the top and bottom part of the automaton respectively. The automaton also captures fault occurrence and repair and replace costs. The severity of faults increases over time. A nozzle may break as soon as two items are painted but it will break for sure once four items are painted. Repairs cost 24 time units (see locations `repairingA` and `repairingM`), replacing a nozzle costs four tokens.

The remaining modelling problem is to describe the alternative assumptions that are being made about users. Two processes are designed to reflect what the user does. The first model (Figure 5) carries out a number of actions. It dispatches conditional user inputs and models simple repair/replace decisions: if the fault (variables p1fault and p2fault) is bigger than 3 and sufficient funds (variable win) are available, replace a nozzle, otherwise perform a repair.

The second user process (Figure 6) implements a random strategy. This process dis-

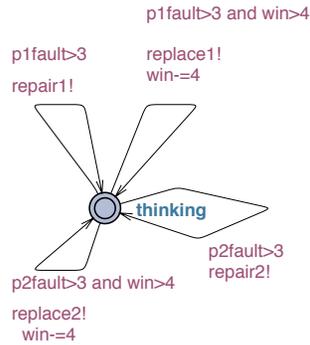


Fig. 5. Process implements a simple repair strategy

patches unconditional user inputs that are consumed by other processes (“monkey at the keyboard” style) but only generated when no internal synchronisations can be performed. Using the two models that have been described it can be used to explore a

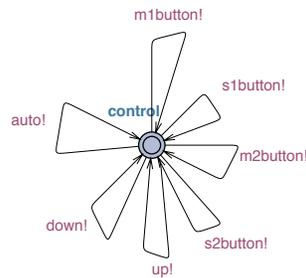


Fig. 6. Process which implements a random user

variety of properties that provide constructive information about the appropriateness of the design of the device.

Reachability of system goals Analysis proceeds experimentally by exploring a number of properties:

P1: Can all n items be painted?

The property (“ $\exists n \text{ painted} = n$ ”) is true for $0 \leq n \leq 10$. When the negated property (here, the never-claim $\forall n \text{ painted} \neq n$ in other words “ n items can never be painted”) is checked, the model checker produces a trace that can be simulated. Stepping through that trace, the analyst is guided through a scenario where both manual and automatic mode of painting are applied. The simulation and the sequence chart provided by uppaal can point to simple flaws or instances of unexpected behaviour of the model. In order to obtain a broader understanding of the reasons behind flaws, additional traces of similar instances are required. However, the tool only produces a single trace for each property. Additional traces, focussing on different aspects that may be considered contributing factors to a discovered problem, require a refinement of the property. For instance:

P2: Can all n items be painted, using only a single paint station?

The verifier only explores paths that involve a single instance of the paint station process. This is achieved by temporarily modifying the paint station specification so that only one of them used.

Finding minimal durations under different conditions Reachability properties may be further elaborated by considering how long it takes to reach a given state. In these cases it is necessary to explore alternative possible times to find the actual duration associated with an activity.

P3: Can all n items be painted in m time units, using only a single paint station?

This can be expressed as “ $\exists n (\text{painted} = n \text{ and } \text{stationUsed} = 0 \text{ and } \text{ptime} = m)$ ” This property was checked for different values m of a global clock ptime . By repeated analysis the property is satisfied for 22 units for the ten items, but the nozzle needs to be replaced at least twice, so the win is only two units. In the same way the following property can be explored:

P4: Can all items be painted in m time units, using both paint stations?

Again, the minimal duration is 22 time units. However, while the execution time is the same, in this case only one of the nozzles needs to be replaced, so the monetary win is six units.

All the traces captured by these properties confirm that the fastest way to perform the work is to opt to paint manually. To consider the design of the automation, the temporal effect of an automatic strategy was considered.

P5: What is the minimal time required to paint all items automatically?

A similar temporary modification to the one described above preventing the manual mode was performed in order to explore this property. The minimal time required to paint all items without manual intervention and by using both stations is 29 units.

Analysis of this kind has yielded the following observations that can be made use of in the design of the interface to the device:

1. Painting items manually is faster than automatic painting.
2. Using both stations does not necessarily gain a time advantage over using a single station only.
3. However, using both stations can save repair costs if the operator is prepared to take the risk and leave one station broken.

The analysis is described more fully in [?]. It should be noted that the temporal properties of this stage could have been calculated simply by using a numeric model of the processes. However, the additional effort of creating the uppaal model pays off when multi-valued decisions are considered, as a focus on monetary costs demonstrates.

Focussing on monetary costs So far the analysis has only been concerned with temporal costs and effects. Further properties can be used to check temporal and monetary costs associated with replacing faulty nozzles.

P6: Can all boxes be painted without losing money?

This property forces a search strategy where nozzle replacements are avoided. The resulting trace demonstrates that the task can be completed in 50 time units. The simulation demonstrates that both stations are used to paint in automatic mode until they break; then one station is repaired.

P7: What is the shortest time for painting everything without losing money?

The analysis yields that best performance (finishing the task in 44 time units) can be achieved, and the new trace suggests that this performance can only be achieved if manual control is opted. Again, both stations break, but the trace indicates that only one station needs to be repaired.

P8: Can all items be painted without losing money, using only one paint station?

This analysis is dual to P6, but focussing on a modified specification so that there is a single paint station only. This property is concerned with the robustness of the system and the additional temporal costs. The strategy exhibited by the model-checking trace could be used by an operator who does not have time pressure and therefore aims at maximising the win.

Analysing the durations under the assumption that temporal costs are secondary to monetary costs reveals again that the best possible performance can be achieved by using both stations in manual mode, but the required duration increases to 44 units. The results produced provide an indication of what a good operation strategy might be under extreme conditions with respect to temporal and monetary costs. However, it remains the task of the system designer to resolve if any of these strategies are suitable, and if they should be implemented as part of the system or as part of the operator training. For an informed decision it also remains necessary to draw on human-factors experience. A crucial additional factor that will influence this decision is the operator workload.

Variable workload The analysis of performance has assumed a constantly high workload, given by the dispatcher model in Figure 2. The analysis can be repeated using increasing, decreasing or alternating workloads in order to collect insights about further strategies.

4.4 An airport ambient system

By way of contrast the second example involves the exploration of an ambient and mobile system as it would appear in a built environment. Here the timescale is at the “minutes” level because it concerns what a passenger would perceive as urgent or immediate in the context of the environment. This system allows access to services, either

global services or services that are specific to the environment (for example, passenger information about travel delays or the status of a flight or retail service information). This information might be invoked and deployed to hand-held devices as well as being displayed in a suitable format on public displays. To get a sense of the style of the system that is to be modelled and analysed consider the following scenario based on an airport.

On entry to the check-in hall, a sensor recognises the passenger's electronic ticket and therefore subscribes her to the flight service for which she is booked. Her context will be updated with current location, namely the check-in hall. The flight service immediately sends information about flight progress to her hand-held that contains queue, gate and delay information. The queue that she is notified about ensures the shortest waiting time. Queue messages are also sent to the public display in the check-in hall so that passengers have an alternative source of information. When the passenger enters the queue a sensor detects her presence and adds this locational information to her context. As a result of this change of location, information about queues are no longer sent to her hand-held device but instead she receives messages that relate to the length of queue and the predicted waiting time. The passenger is meanwhile subscribed to a seat booking service specific to the flight that enables her to book the seat she likes while waiting. The service helps the passenger choose based on her preference information.

Properties of concern would depend on a requirements elicitation but would feasibly include examples such as: "The system should ensure that flight information relevant to all passengers contained within a space is supplied by posting that information on the displays of that space". In general required properties would concern: the resilience of the whole system, including temporal characteristics; the usability of the hand held devices and public displays; how effectively and immediately information is provided that enhances the experience of users of the system. In the last case timing issues may be critical to achieving this experience and a sense of place in the built environment. Properties of concern here will include "however many services a user subscribes or is subscribed to, the flight information service will be dispatched both to the user's device and to the local display within a defined time interval" and "any facility that is offered to a subscriber will only be offered if there is a high probability that there is enough time to do something about the facility offered."

In reality the development of ambient mobile intelligence applications might be carried out without complete foreknowledge of the platforms and software that will be running on these platforms. The physical characteristics of alternative platforms may be important in contributing to the experience of place — frequent flyers may use smart phones, large plasma screens may be placed in the space in a number of different ways. The advantage of using walkthrough techniques is that early exploration may be carried out before the platform is decided and may assist an understanding of whether a particular configuration is appropriate.

4.5 Modelling the airport

To model this system some gross simplifications are made to indicate the style of analysis. The description captures the characteristics of the airport system by modelling sensors for each space in the built environment, the single dispatcher that is designed to

distribute messages and a token passenger that captures the properties of all the passengers in the environment.

The dispatcher (Figure 7) simply distributes messages with tags associated with flight number and location. In the version described here it distributes these at random at different time intervals depending on the value of workload. The rate of distribution can be adjusted to assess the properties of different rates of distribution.

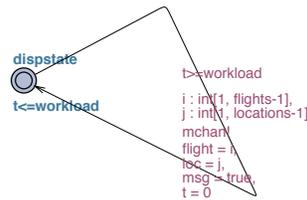


Fig. 7. The dispatcher process

The distributed characteristics of the airport system are captured by three further models. Two of these models represent different types of sensor, one relates to the queue in the checkin area while the other is a more generic sensor (Figure 8) that captures the location of the different areas of the airport while at the same time receiving messages from the distributor. These messages are filtered according to location. They are marked as displayed on the public display if the sensor has received requests from passengers with flight details that match the flight tag. In other words the public display is modelled so that it only displays those messages that are relevant to the passengers in the area.

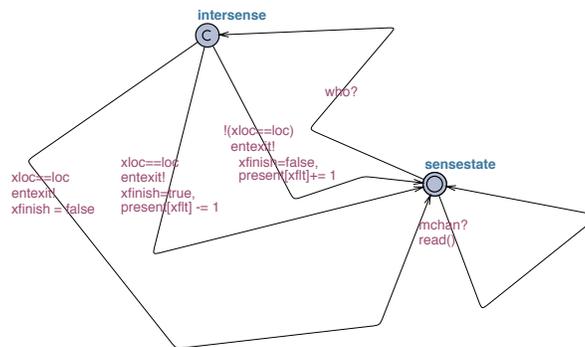


Fig. 8. The non-queueing sensor

The other sensor is designed to updates its display with information about its queue based on the longest wait. The non-queueing sensor checks the location and status of

a passenger that communicates with it. Depending on the flight number and whether the passenger has just entered the location or is about to exit it, the count for the flight is decremented or incremented. When a message is received from the dispatcher for a particular flight it is only dispatched if it is valid for the location and there are passengers in the location that relate to the flight — this is done by the `read()` function.

The passenger (Figure 9) checks its location by communicating its flight number to the sensor. When the passenger initialises it absorbs a strategy for navigating through the building by simply updating an array `path`. It receives messages from the dispatcher and displays them on the hand-held only if the location and flight match the passenger in the distributed message.

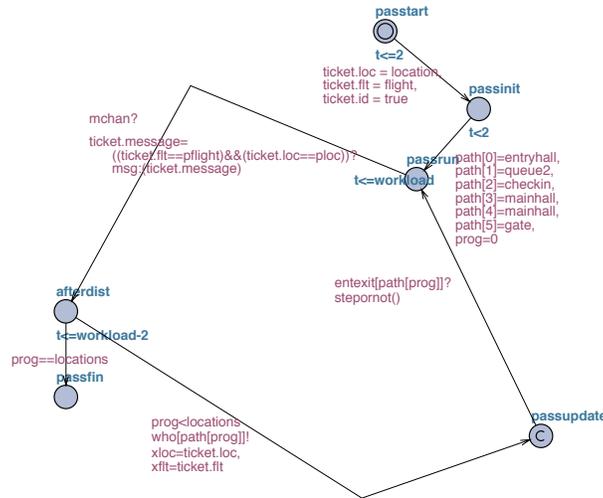


Fig. 9. The passenger

5 Conclusion

This chapter has identified features in the design of an interactive system that may be influenced by timing characteristics and has provided a preliminary illustration of how modelling techniques might assist the exploration of the design. The chapter contains a challenge — features of a design in response to timing considerations are rarely considered and yet have an important impact on the usability and the experience of the system design. Within the context of time orientated design it makes a number of observations, for example the fact that temporal issues may be external, a feature of the environment, or may be internal and perceived. It may therefore be necessary to explore external cognitive resources which the user can offload to so that perceived workload may be reduced. We have demonstrated that while modelling techniques that are famil-

iar to computer scientists can be used there are a number of challenges that must be overcome before these techniques become practically feasible.

1. dealing with the state explosion that is associated with adding temporal constraints
2. in the case of the ambient and mobile systems, dealing with the fact that there may be many processes with similar properties (such as passengers) that must be incorporated within the model without exploding the specification so that it is impractical
3. finding appropriate languages or patterns of use of the modelling techniques to capture notions of location and context in a way that makes it easier for implementers to use these techniques without serious and unnecessary overhead.

References

1. Conn, A.: Time affordances: the time factor in diagnostic usability heuristics. In: Proceedings of the SIGCHI conference on human factors in computing systems, ACM Press (1995) 186–193
2. Hollnagel, E.: Cognitive Reliability and Error Analysis Method – CREAM. Elsevier (1998)
3. Wright, P., Fields, R., Harrison, M.: Analyzing human-computer interaction as distributed cognition: the resources model. *Human-Computer Interaction* **15** (2000) 1–42
4. Campos, J., Doherty, G.: Supporting resource-based analysis of task information needs. In Harrison, M., Gilroy, S., eds.: Proceedings 12th International Workshop on the Design, Specification and Verification of Interactive Systems, Springer Lecture Notes in Computer Science (2006) in press.
5. Campos, J., Harrison, M.: Model checking interactor specifications. *Automated Software Engineering* **8** (2001) 275–310
6. Loer, K., Harrison, M.: An integrated framework for the analysis of dependable interactive systems (ifadis): its tool support and evaluation. *Automated Software Engineering* (2006) in press.
7. Loer, K., Harrison, M.: Analysing user confusion in context aware mobile applications. In Constabile, M., Paternò, F., eds.: INTERACT 2005. Number 3585, Springer Lecture Notes in Computer Science (2005) 184–197
8. Nielsen, J., Mack, R., eds.: Usability Inspection Methods. John Wiley & Sons, Inc. (1994)
9. Hildebrandt, M., Harrison, M.: Putting time (back) into dynamic function allocation. In: Proceedings of the 47th Annual Meeting of the Human Factors and Ergonomics Society, Denver, CO (2003) 488–492
10. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer* **1** (1997) 134–152

Connecting rigorous system analysis to experience centred design

M. D. Harrison¹, J. Creissac Campos², G. Doherty³, and K. Loer⁴

¹ Informatics Research Institute, University of Newcastle upon Tyne, NE1 7RU, UK
Michael.Harrison@ncl.ac.uk

² Department of Informatics, Universidade do Minho, Campus de Gualtar 4710-057 Braga, Portugal
Jose.Campos@di.uminho.pt

³ Dept of Computer Science, Trinity College Dublin, Ireland
Gavin.Doherty@cs.tcd.ie

⁴ Department Strategic Development, Germanischer Lloyd AG - Head Office, Vorsetzen 35,20459 Hamburg, Germany
Karsten.Loer@gl-group.com

Abstract The chapter explores the role that formal modelling may play in aiding the visualisation and implementation of experience requirements in an ambient and mobile system. Mechanisms for requirements elicitation and evaluation are discussed, as well as the role of scenarios and their limitations in capturing experience requirements. The chapter then discusses the role of formal modelling by revisiting an analysis based on an exploration of traditional usability requirements before moving on to consider requirements more appropriate to a built environment. The role of modelling within the development process is re-examined by looking at how models may incorporate knowledge relating to user experience and how the results of the analysis of such models may be exploited by human factors and domain experts in their consideration of user experience issues.

1 Introduction

Ambient and mobile systems are often used to bring information and services to the users of complex built environments such as leisure complexes, hospitals, airports and museums. The success of these systems is dependent on how users *experience* the space in which they are situated. They may serve to provide users with an experience of the built environment as a place rather than a forbidding sterile space. They may serve to alleviate the anxiety of travelling in an unfamiliar world. The problem of concern in this chapter is how to reason about such systems so that they satisfy experience requirements.

A focus on experience and ambient and mobile systems provides an important trigger for a fresh look at evaluation in interactive systems. Traditional notions of usability need reconsideration. Ambient and mobile systems have distinctive characteristics that lead to a requirement for special treatment:

- the impact of the environment as the major contributor in understanding how the system should work — its texture and complexity

- the possible role of location and other features of context in inferring user action implicit or incidental in the activities of the user — how natural and transparent this inference is.

For these reasons it is difficult to assess ambient and mobile systems early in the design process. The chapter explores how experience requirements can be related to more rigorous methods of software development. The chapter's structure takes phases of the design process to explore how experience requirements can be implemented within a system.

The chapter begins by discussing how experience requirements can be elicited from an existing system (Section 2). It then moves onto the means of assessing and evaluating a proposed design against such requirements (Section 3). It considers ways in which these requirements can be explored effectively using formal techniques: by considering interface requirements within a process control system (Section 4); and then considering experience requirements more explicitly by speculating about information flows (Section 5) in an airport system. Finally the chapter sketches a future agenda for completing the objectives established.

2 Eliciting and making sense of user experience requirements

A conclusion that may be drawn from the challenges mentioned in the previous section is that the evaluation of such ambient and mobile systems must be carried out in-situ within the target environment, with typical users pursuing typical activities. The problem with this conclusion is that it is usually infeasible to explore the role of a prototype system in this way, particularly when failure of the system might have safety or commercial consequences. Methods are needed therefore that would enable the establishing of experience requirements and to explore whether they are true of a system design before expensive decisions are made.

Eliciting experience requirements for an envisaged ambient system can be carried out using a combination of techniques. For example, stories can be gathered about the current system, capturing a variety of experiences, both normal and extreme, and visualising the experiences that different types of user or persona might have in the design. The results of this story gathering process will be a collection of scenarios that can be used to explore how the new design would behave. They can be used to evaluate the design (see for example [Rosson and Carroll, 2002]), perhaps using a specification of the design or using a rapidly developed prototype. For more traditional usability requirements, techniques such as cognitive walkthrough [Lewis et al., 1990] or co-operative evaluation [Monk et al., 1993], can provide valuable complementary approaches to evaluation based on these scenarios. In addition to scenario orientated techniques for elicitation other techniques are also valuable. Techniques such as cultural probes [Gaver et al., 1999] can be used to elicit "snapshot experiences". The elicitation process here involves subjects collecting material: photographs, notes, sound recordings to capture important features of their environment. While these snippets may make sense as part of a story they may equally well be aspects of the current system that are common across a range of experiences or stories.

A question then is how to make sense of these snapshots. Consider the example of a frequent flyer who is nevertheless anxious about missing his flight. One could imagine that he might take a snapshot of the public display and comment that he always looks for a seat where this information is visible. He might also comment: that the flight information relevant to him is not always clearly discernible on the display; that delay information is often displayed late and is not updated so there is no sense of there being any progress. This information is not captured well by a specific scenario because, although one such situation can be captured well, the scenario does not cover all situations — that this information needs to be available whatever “path” the user takes. Alternative approaches are required that will enable the exploration of all possible user paths.

[Buchenau and Suri, 2000] describe an approach they call “experience centred design” that involves the construction of prototypes, sometimes very inexpensive and approximate prototypes, which can be used to imagine the experience that users would have with the design. The quality and detail of the prototype tends to vary: from “mocking up” using non-functional prototypes to more detailed prototypes that are closer to the final system. To explore and to visualise the proposed design effectively, it is important that systems can be developed with agility, using a context that is close to the proposed target environment. They help envision the role of the “to-be-developed” artefact within that work. Prototypes can be used to “probe”, that is explore, the validity and representativeness of the scenarios and may lead to alternative or additional scenarios. Testing the prototypes appropriately can develop an understanding of the experience of the system in its proposed setting.

Consider, for example, a system developed to help passengers experience a sense of place in the unfamiliar setting of an airport. One might imagine a combination of ambient displays, kiosks and mobile services for hand-held devices. They combine together to provide an environment in which passengers can obtain the information they need, in a form that they can use it, to experience the place. Information about the environment relevant to an understanding of this experience might be captured using a combination of cultural probes and scenario analysis. For example in the case of cultural probes, passengers might be asked to identify those elements in the space that relate strongly to their experience of the airport, perhaps by taking photographs or making audio-video recordings and then by annotating these snapshots. In addition they might be asked to tell stories about situations where they did or did not experience place. The following examples might derive from such elicitation:

- photographs of the main display board with comments such as: “I like to be in a seat in which I can see this display board”; “I wish that the display board would tell me something about my flight — it disturbs me when it simply says wait in lounge”;
- photographs of signposts pointing to where my gate is annotated with “I wish I had better information about how far it was and whether there were likely to be any delays on the way”;
- tape recordings of helpful announcements and tape recordings of unhelpful announcements, with annotations such as “These announcements do not happen often enough and announcements for other flights distract me”;

- stories about where the airport helped me to feel aware of what was happening;
- stories of long and complicated situations that caused me problems.

Thus an idea can be obtained about how the system works. It becomes possible to capture stories that deal exclusively with subsets of the required functionality, for example one might deal with details of flights, whether there is food, what the reason for the delay is, whether there are any other flights that I can catch that would get me to Los Angeles today. Another story might relate to whether there is enough time to get a meal and whether the meal is vegetarian. Hence in parallel, prototypes might be developed that deal with segmented functionality — a prototype dealing with flights and flight schedules; a prototype dealing with retail services. Prototypes might be explored, running in-situ using the user stories as the means of testing, exploring the prototype in a simulation of the situation, assessing whether an experience of place is being contributed to. This means that the whole system might be built up using partial prototypes thereby reducing the need to wait until a complete system is available.

There is a problem with scenarios however they are elicited. They cannot capture all aspects of the experience of place in the airport. The value of cultural probes on the other hand is that they provide an orthogonal viewpoint. In order to achieve an experience of place, the familiar things – for example the constant presence of the notice board – must be captured across scenarios. It is not sufficient simply to focus on scenarios in order to establish a proper sense of the overall experience of the envisaged system under design. Further exploration may be required to assess and probe how well these static elements of the environment (such as the continually present notice board) are represented across all possible behaviours of the design. It is also necessary to investigate the unforeseen consequences of the proposed design. The complexity and interaction between the different components of the system may result in unexpected, emergent properties of behaviours. As a system design evolves, so will the experience associated with using the system. This can contribute to producing a more consistent overall experience, even though the design of the system has emerged in piecemeal fashion.

The physical characteristics of alternative platforms may be important in contributing to the experience of sense of place — frequent flyers may use smart phones, large plasma screens may be placed in the space in a number of different ways. The advantage of using walkthrough techniques is that early exploration may be carried out before the platform is decided and may assist an understanding of whether a particular combination of system components is appropriate.

In the next section evaluation and analysis techniques shall be considered in more detail. Before doing this two short descriptions of how an airport system might work will set the context more concretely.

- On entry to the departures hall, a sensor recognises the electronic ticket and subscribes the passenger to the appropriate flight while updating the passenger's context to include current position in the departures hall. The flight service publishes information about the status and identity of queues for check in. A message directing the passenger to the optimal queue is received by the passenger's hand-held device because the passenger's context filter contained in the device permits its arrival. This information is displayed on a public display in the departures hall. When the passenger enters the queue a sensor detects entry and adds the queue identifier

- to the passenger information. As a result different messages about the flight are received by the passenger — this might include information about seating so that the passenger can choose a seat while waiting to check in baggage. This process continues as the passenger progresses through the various stages of embarkation.
- The passenger enters the main hall. The passenger is now additionally subscribed to a retail service. Information about available facilities are received by the passenger according to preferences and flight status.

The software framework that is implicit here is based on a publish-subscribe system [Eugster et al., 2003].

3 Analysis and evaluation

[McCarthy and Wright, 2004] have argued that while the emphasis within the GUI paradigm has been on technology as tools, the new paradigms require thought about technology we live with (see also [Bannon, 2005]). Elsewhere, this has been characterised as a shift from understanding the use of artefacts to understanding their presence in people’s lives [Halnass and Redstrom, 2002]. While user-centred design helps understand the practices and routines into which technologies are expected to fit, they are not as helpful with feelings of resistance, engagement, identification, disorientation, and dislocation. Prototypes can be explored from a variety of perspectives, from a spectrum of usability-engineering evaluation techniques to “experience” explorations through active engagement with prototypes (see [Buchenau and Suri, 2000] and [IST, 2004]). Techniques that are used should be formative and prototypes developed within the simulated scene may be used to stimulate communication and exploration of design ideas as a dialogical process between user, designer and software engineer. A number of techniques may be used to identify experience characteristics of a design.

3.1 Scenario analysis

Scenario analysis [Rosson and Carroll, 2002] can be used at a number of levels to explore the role that the system might play and to evaluate usability and experience issues. Scenarios can be used to capture important characteristics of the environment, either typical uses of the system or “critical incidents” where current arrangements have failed users. They can be analysed by usability engineers to explore how the system would work — what information would be displayed at specific times within the scenario, what actions the user would have to take to obtain further information and so on. Techniques such as cognitive walkthrough [Lewis et al., 1990] and THEA [Pocock et al., 2001] are designed to be used at the action level by usability engineers who have enough knowledge of the environment or a sufficiently detailed scenario to be able to consider and visualise the design. While reservations are appropriate in terms of their objectivity [Gray and Salzman, 1998] they are nevertheless of value as a formative mechanism in the hands of designers.

Scenarios can also be “visualised” by users as they re-experience in their imaginations the scenario in the context of the new design. This might involve the user adopting

a persona – a frequent flyer who is nevertheless an anxious flyer. This would not create a detailed account of how the technology works rather it would provide an impression of aspects that require further analysis. Assessing how an artefact contributes to experience requires observation or assessment of the artefact embedded in the proposed situation. Although experience prototypes can be constructed, simulated conditions are required that can deal with realistic scenarios in order that a “passenger-to-be” might visualise the effect that the proposed technology would have and how it would feel to use it. Consider, for example, a system developed to help passengers experience a sense of place at check-in, security screening, passport control and while waiting in the main body of the airport and making use of the many facilities made available to them.

3.2 Alternatives to scenario analysis

Scenario analysis inevitably restricts consideration of the system to particular situations and therefore issues of coverage are important. Often experience requirements lead to properties that hold true whatever the situation. Experience level requirements that can be captured specifically for the application in question can be dealt with in a similar way to usability inspection where properties are checked systematically by a team using heuristics [Nielsen, 1992].

[Campos and Harrison, 2001] and [Loer and Harrison, 2006] explore the synergistic role that modelling and scenario based evaluation can play. Properties, formal expressions of usability heuristics, are used to generate traces, that is sequences of actions in the model that serve to demonstrate a situation where the property does not hold. These traces can provide the basis for scenarios. Domain experts can use the bare sequence of actions to create a plausible narrative that can form the basis of a scenario. This scenario can then be subjected to an analysis such as a cognitive walkthrough in order to explore potential problems with the interface to the design. Consider an example of mode confusion. A system is checked for some formal representation of mode confusion and a trace is generated that indicates a circumstance where confusion might occur. This forms the basis for a scenario that is investigated. It is quite possible that although formally there is mode confusion, the interface signals the mode so clearly that in practice it will not be a problem. This kind of analysis can also be carried out for properties that result from an exploration of the experience requirements of the design. Suppose that a passenger reports that she wants to be able to access up to date flight information wherever she is. An appropriate model might be used to explore possible paths that passengers might take to reach the flight gate and whether up-to-date flight information is always available. This approach is analogous to that taken in [Loer and Harrison, 2005] where a system is explored that controls a process either using a central control room or a hand-held PDA. This will be explored in more detail in the next section.

3.3 Modelling

Formal modelling techniques and agile software development may both have contributions to make to experience centred design. The modelling approach provides the basis for exploring paths that are used by domain experts or usability experts to create narratives that can then be used to explore the experience. Viewing a design in terms

There may be goal relevant properties, to support users in achieving their goals in those spaces: “when the traveller is late in carrying out the next action, for example screening baggage, a message will be sent to the traveller’s hand-held informing them of the urgency of their need to move to the next stage”. In reality the development of ambient mobile intelligence applications might be carried out without complete fore-knowledge of the platforms and software that will be running on these platforms. The advantage of using walkthrough techniques is that early exploration may be carried out before the platform is decided and may assist an understanding of whether a particular configuration is appropriate. However it may also be feasible to use formal approaches to provide this analysis and thereby make some of the processes more tool supported and systematic as explored in [Loer and Harrison, 2006].

4 Properties of interactive devices

The external world can be explored in terms of how it provides resources for the actions available through the device. [Campos and Doherty, 2006] have done preliminary work concerned with modelling and investigating the interaction between devices and users — exploring whether, given a set of (dynamically) available resources (in some place, and at some point in time), users will be able to achieve certain goals.

This section diverges from the airport example to demonstrate the process in a more fully worked example designed to explore conventional usability requirements of a mobile system [Loer and Harrison, 2005]. It is concerned with goal related properties of a mobile device within a process plant. It concerns the operator interface to a process control system from a centralised control room (see Figure 2) as well as an alternative hand-held device (see Figure 3) [Nilsson et al., 2000]. A limited subset of information and controls for these components will be “stored” in the hand-held device to ease access to them in the future – analogous to putting them on the desktop. These desktop spaces are called *buckets* in [Nilsson et al., 2000]. The operator can view and control the current state of the components when in their immediate vicinity. Context is used in identifying position of an operator, checking validity of a given action, inferring an operator’s intention, checking action against an operator’s schedule, while assessing and indicating the urgency of these actions.

In this type of system, context confusions can be avoided through design by changing the action structure (for example, using interlocks) so that these ambiguities are avoided or by clearly marking the differences to users. Techniques are required that will enable the designer to recognise and consider situations where there are likely to be problems. However there is a further issue, namely that the experience of the device and the system by the user might support safety more or less effectively. In the analysis that proceeded, the exploratory approach described in the previous section was used to scrutinise traces that are “interesting”. Implications of different configurations are explored by considering simple assumptions about the user. An analysis is now described in which questions are articulated in LTL (Linear Temporal Logic) and recognised by the SMV model checker [McMillan, 1993]. There are no details of the specifications here — these can be found in [Loer and Harrison, 2006].

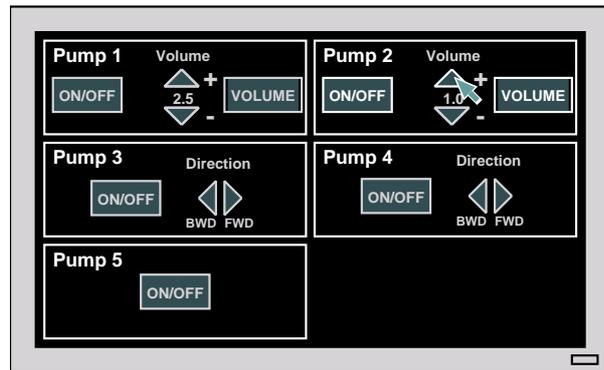


Figure 2. Control Screen layout.

Both the hand-held device and the control room were modelled using Statecharts [Harel, 1987] as was the plant. A requirements process might plausibly have generated the following top level requirements for the interactive system controlling the plant: (1) to inform the operator about progress; (2) to allow the operator to intervene appropriately to control the process; (3) to alert the operator to alarming conditions in the plant and (4) to enable recovery from these conditions.

The plant involves tanks, pipes, valves and pumps that feed material between tanks. The tanks are designed to be used for more than one process and, in order to change processes, a tank must be evacuated before material can be pumped into it. In order to achieve this some of the pumps are bi-directional. The functioning of the plant, the flows and evacuations can be expressed as a simple discrete model so that the significant features of the environment can be explored. This is discussed in more detail in [Loer and Harrison, 2006]. The plant is defined in terms that provide the simplest way in which the control interface is aware of its function. Hence the state of the tank is simply described as one element of the set $\{full, empty, holding\}$ — there is no notion of quantity or volume in the model.

The control room, with its central panel, aims to provide the plant operator with a comprehensive overview of the status of all devices in the plant. Situation awareness is considered to be critical to the operator's work in the system — in experience terms the operator needs to know that they can see everything that is going on. Availability and visibility of action are therefore seen to be primary concerns. For this reason a model of the interface is chosen that focuses on these aspects of the design. Other models could also have been considered to focus on other facets, for example alarms or recoverability. The control panel is implemented by a mouse-controlled screen (see Figure 2). Screen icons are both displays and controls at the same time — clicking on an icon will have an effect. These features of the design are all modelled, showing when icons are illuminated and when actions trigger corresponding actions in the underlying process. The Statechart here builds a bridge between actions that relate to the behaviour of the process underneath and actions performed by the user, such as using the mouse to point and click at the relevant icons.

The hand-held device uses individual controls that are identical to the central control panel. However there is only a limited amount of space available for them. As a controller walks past a pump she may “save” controls onto the display. While the controls continue to be visible on the display, the pumps relating to the controls can be manipulated from anywhere in the system. The hand-held control device (Figure 3) knows its position within the spatial organisation of the plant.

By pointing a “laser pointer” at a plant component and pressing the component selector button, the status information for that component and its controls are transferred into the currently selected bucket. Components can be removed from a bucket by pressing the delete button. With the bucket selector button the user can cycle through buckets. The specification of the hand-held device describes both the physical buttons that are accessible continuously and other control elements, like pump control icons, that are available temporarily and depend on the position of the device. When the operator approaches a pump, its controls are automatically displayed on the screen (it does not require the laser pointer). The component may be “transferred” into a bucket for future remote access by using the component selector button. Controls for plant devices in locations other than the current one can be accessed remotely if they have been previously stored in a bucket. When a plant component is available in a bucket and the bucket is selected, the hand-held device can transmit commands to the processing plant, using the pump control icons.

In the case of the hand-held control device the interface to be explored is the device in the context of its environment. The environment in this case is a composition of the tank content model and the device position model. The model presumes that the appliance should always know its location. An alternative approach would allow the designer to explore interaction issues when there is a dissonance between the states of the device and its location. The effect of the type of software architecture used to implement these types of system is to mask the possibility of discrepancy from the implementer.

In order to explore the effect of the difference between the control room and the hand-held device and to generate traces that may be of interest a reachability property is formulated for a user level “goal” of the system. The goal chosen here for illustration is “*Produce substance C*”. This is a primary purpose of the system. The analysis

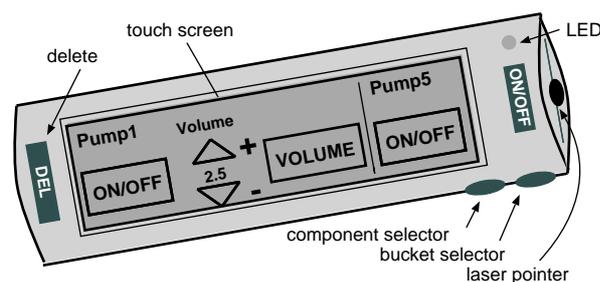


Figure 3. A hand-held control device (modified version of the “Pucketizer” device in [Nilsson et al., 2000]).

proceeds by making a comparison between traces generated by the alternative models, using domain knowledge and user experience to generate appropriate scenarios. If a property does not hold then the checker finds one counter-example. Alternatively, the negated property may be used to find a trace that satisfies the property. Usually the model checker only produces a single trace giving no guarantee that it is an interesting one from the point of view of understanding design implications. Additional traces can be created by adding assumptions about the behaviour. This contrasts with an approach using explicit tasks (see for example, [Fields, 2001,Loer, 2003]) where the model checker is used to explore a particular way in which the goal can be achieved (the task). So far as this chapter is concerned any behaviours required to achieve a goal are of interest.

The sequences in Figure 4 represent the traces obtained by checking for different models if and how the plant can deliver substance *C* to the outside world. The property asserts that, eventually, pump 5 will be turned on with tank 1 holding substance *C*. The two models involving the different interfaces are checked with the same property. These sequences can be used to provide the basis for scenarios that would enable a domain expert and human factors expert to assess the interaction. One possibility would be that a narrative is described around the sequence and this narrative used by a potential operator to visualise the experience that they would have using the designed system, much as scenarios and snapshots elicited from the current systems are used. The first sequence in Figure 4 satisfies the control room interface. The second sequence was generated by checking the property against the hand-held device model. While the first two traces assume a serial use of pumps, the third and fourth sequences show the same task for a concurrent use of pumps. Comparison of these sequences yields information about the additional steps that have to be performed to achieve the same goal.

As a result of making a comparison between the traces for the control room and for the hand-held, the analyst might come to the conclusion that the repetitive process of saving controls may cause slips or mistakes, a direct effect of location on the actions of the hand-held device. While these slips or mistakes may not be dangerous, it may be concluded that the frustration of continually delaying because of omitting actions may be significant and therefore the experience of the design will be affected negatively. To explore the effect of this a further assumption may be introduced to the property to be analysed, namely that an operator might forget certain steps.

For example, if it is assumed that controls for the pumps are not saved and the original property is checked, the sixth sequence in Figure 4 is obtained. This sequence highlights the likelihood of context confusions as well as user frustrations and therefore the need for the redesign of the device. As can be seen, an identical subsequence of actions at positions 2 and 6 have different effects. An interlock mechanism might therefore be introduced to reduce the frustration caused by forgetfulness. The proposed redesign warns the user and asks for acknowledgement that the currently displayed control elements are about to disappear. The warning is issued whenever a device position is left and the device's control elements are neither on screen nor stored in a bucket. It is straightforward to adjust the model of the interface to the hand-held device to capture this idea, and this specification is given in [Loer and Harrison, 2004]. The design however does not prevent the user from acknowledging and then doing nothing about the

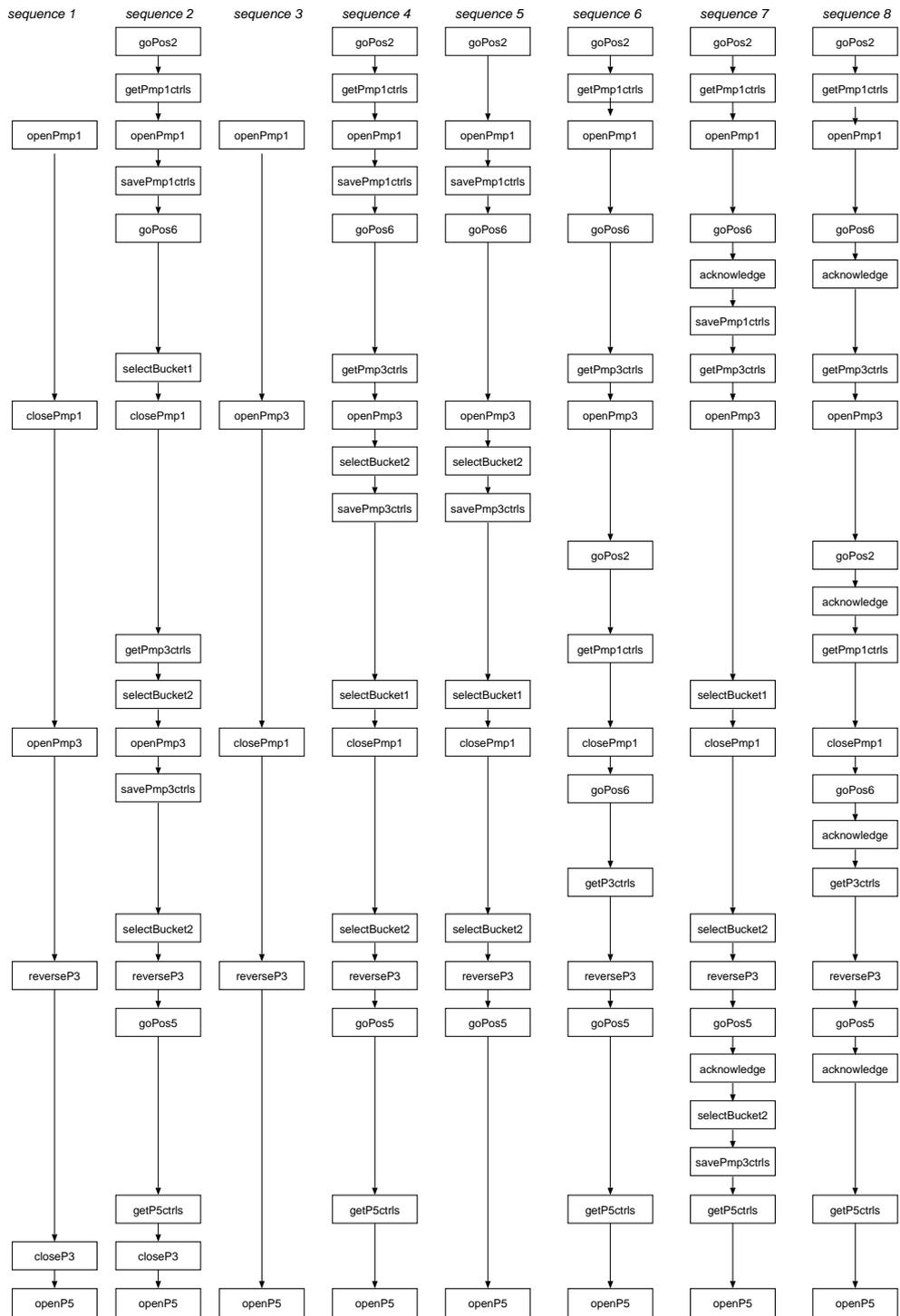


Figure 4. Traces generated by runs of the model checker

problem. Checking the same properties, including the assumptions about the forgetful user, produces Sequences 7 and 8 in Figure 4. In this example the central control panel can be used to identify the key actions to achieving the goal since the additional actions introduced by the hand-held device are concerned exclusively with the limitations that the new platform introduces, dealing with physical location, uploading and storing controls of the visited devices as appropriate. The analysis highlights these additional steps to allow the analyst to judge if such additional steps are likely to be problematic from a human factors perspective. The reasons why a given sequence of actions might be problematic may not be evident from the trace but it provides an important representation that allows a human factors or domain analyst to consider these issues. For example some actions might involve a lengthy walk through the plant, while some actions may be performed instantaneously and some might depend on additional contextual factors like network quality. The current approach leaves the judgement of the severity of such scenarios to the designer, the human factors expert or the domain expert. It makes it possible for these experts to draw important considerations to the designer's attention.

5 Information arrival

In the previous section an analysis performed on a control system to explore usability properties was used to demonstrate the proposed formal approach. Although, no experience requirements elicitation has been performed on that system it was possible to speculate on requirements associated with frustration manifested in the necessity to repeat actions unnecessarily. The airport example is considered in a little more detail to illustrate the analysis of provision of information to the user (space does not permit a detailed analysis). Properties to be described in this section may be analysed using models of the type described by [Garlan et al., 2003] and [Baresi et al., 2005]. With such models it is possible to explore the integration of sensors and appropriate ways to introduce filters associated with context using a publish-subscribe architecture, enabling the exploration of properties such as:

- when the passenger enters a new location, the sensor detects the passenger's presence and the next message received concerns flight information and updates the passenger's hand-held device with information relevant to the passenger's position and stage in the embarkation process.
- when the passenger moves into a new location then if the passenger is the first from that flight to enter that location, public displays in the location are updated to include this flight information
- when the last passenger on a particular flight in the location leaves it then the public display is updated to remove this flight information
- as soon as a queue sensor receives information about a passenger entering a queue then queue information on the public display will be updated.

These properties can all be related to the experience that a user has of the system. The system's failure to adhere to all of these properties will not necessarily mean that the system cannot perform effectively but in some sense or other they may relate to the potential for anxiety or a sense of where the passenger is. Given the style of approach

discussed in the previous section, checking properties of the model will generate sequences that do not satisfy them. The domain expert will use this information to generate scenarios that are potentially interesting from a user point of view. These scenarios may then be used perhaps to visualise how different personae would experience them. A potential user might be asked to adopt the persona and then to visualise the system. Paper or electronic prototypes would be used to indicate what the system would appear to be like at the different stages of the scenario.

Many characteristics of systems associated with timeliness or likelihood of occurrence contribute to the experience that we have of them. Such properties require models that incorporate notions of time (the message relating to the flight will be received within a fixed time span) and stochastic models (with a given probability). [Loer et al., 2004] have used `uppaal` models to analyse human scheduling behaviour in relation to process control systems. [Doherty et al., 2001] have explored stochastic properties of interactive systems and [ten Beek et al., 2006] have used both timed model checking and stochastic model checking to analyse a “groupware system”. Properties that are relevant here relate to the dispatching of messages, for example:

1. the message is the next message
2. the message is most likely to be the next message [De Nicola et al., 2005]
3. the message will arrive within 30 seconds [Loer et al., 2004]

Hence further properties of the airport system with an impact on user experience would include:

- no matter how many services a user is subscribed to, the flight information service will be dispatched both to the user’s device and to the local display within a defined time interval
- any service that is offered to a subscriber will only be offered if there is a high probability that there is enough time to do something about the service offered
- when the passenger moves into the location then flight status information is presented to the passenger’s hand-held device with 30 seconds
- information on public displays should reflect the current state of the system within a time granularity of 30 seconds
- if the passenger enters a location then the passenger’s trail will be updated with the action that should occur at that stage (for example screening hand baggage) within an appropriate time (two minutes). If not a reminder of the current activity will be delivered to the user’s hand-held
- queue information relating to the best queue to join for a specific flight will be designed to avoid jitter, that is it will be updated sufficiently frequently to improve the experience of passengers but not so frequently that which queue to join and information about how long the delay in the queue changes in a way that is annoying to passengers.

It can be seen that properties such as these will be particularly appropriate to meet passenger uncertainties about flight status, avoid the frustration of jittering information about queues and of being offered services that cannot be received through lack of time.

6 Conclusions

Ambient and mobile systems provide a rich context for the process of requirements elicitation. They challenge our presumptions about how to analyse interactive systems. A particularly interesting class of such systems provides the occupants of built environments with a sense of the space — to support a feeling of place and provide access to the services that are offered within the environment. The evaluation of the effectiveness of these systems requires the full richness of the target environment and yet in reality it is not possible for a variety of reasons to explore these systems in a live environment. The possibility that these systems can be explored through a process that involves the use of formal methods has been discussed. Part of this has been demonstrated by reconsidering an analysis that was performed with a more traditional usability perspective. Further examples derived from the specific concerns of an ambient and mobile system in an airport environment.

Formal techniques that can be used to capture abstractly the key features of the prototype currently being developed and can be used as a means of simulation or exhaustive path checking. The model can be developed at the same time as the prototype. Using the model it becomes possible to capture, for example, the knowledge that users in the environment might have [Fagin et al., 2004] or the resources for action that are required by users [Campos and Doherty, 2006]. The development of prototypes that support a subset of functions may be accompanied by simple models and simulations in which these prototypes can be explored. So for example, separate models can be developed to reason about the features pertaining to movement through space, and the actions that the user may perform explicitly using the system. Analysis by simulation or model checking can lead to the discovery and exploration of paths that were not envisaged in the original set of scenarios. With the help of domain experts, situations can be envisaged in which the design fails to provide the passenger with the information they need to experience place.

Two important issues underpin our agenda for future research. The first concerns the mapping between models and prototypes and how to maintain an agile approach to the development of prototypes, while at the same time providing the means to explore early versions of the system using formal models. Our concern is to produce generic models that reflect the software architecture used for rapid development and to maintain synchrony between prototype and model. The second concerns the class of models required to analyse the range of requirements that would be relevant to ambient and mobile systems — how to ensure practical consistency between them, and to avoid bias and inappropriate focus as a result of modelling simplifications.

References

- [Bannon, 2005] Bannon, L. (2005). A human-centred perspective on interaction design. In Pirhonen, A., Isomäki, H., Roast, C., and Saariluoma, P., editors, *Future Interaction Design*, pages 31–52. Springer-Verlag.
- [Baresi et al., 2005] Baresi, L., Ghezzi, C., and Zanolin, L. (2005). Modeling and validation of publish / subscribe architectures. In Beydeda, S. and Gruhn, V., editors, *Testing Commercial-off-the-shelf Components and Systems*, pages 273–292. Springer-Verlag.

- [Buchenau and Suri, 2000] Buchenau, M. and Suri, J. (2000). Experience prototyping. In *Proceedings Designing Interactive Systems (DIS00)*, pages 424–433, Brooklyn, New York. ACM Press.
- [Campos and Doherty, 2006] Campos, J. and Doherty, G. (2006). Supporting resource-based analysis of task information needs. In Harrison, M. and Gilroy, S., editors, *Proceedings 12th International Workshop on the Design, Specification and Verification of Interactive Systems*. Springer Lecture Notes in Computer Science. in press.
- [Campos and Harrison, 2001] Campos, J. and Harrison, M. (2001). Model checking interactor specifications. *Automated Software Engineering*, 8:275–310.
- [De Nicola et al., 2005] De Nicola, R., Latella, D., and Massink, M. (2005). Formal modelling and quantitative analysis of KLAIM-based mobile systems. In Haddad, H., Liebrock, L., Omicini, A., R. Wainwright, Palakal, M., Wilds, M., and Clausen, H., editors, *Applied Computing 2005: Proceedings of the 20th Annual ACM Symposium on Applied Computing*, pages 428–435.
- [Doherty et al., 2001] Doherty, G., Massink, M., and Faconti, G. (2001). Using hybrid automata to support human factors analysis in a critical system. *Journal of Formal Methods in System Design*, 19(2):143–164.
- [Eugster et al., 2003] Eugster, P., Felber, P., Gerraoui, R., and Kermarrec, A. M. (2003). The many faces of publish subscribe. *ACM Computing Surveys*, 35(2):114–131.
- [Fagin et al., 2004] Fagin, R., Halpern, J., Moses, Y., and Vardi, M. (2004). *Reasoning about Knowledge*. MIT Press.
- [Fields, 2001] Fields, R. (2001). *Analysis of erroneous actions in the design of critical systems*. PhD thesis, Department of Computer Science, University of York, Heslington, York, YO10 5DD.
- [Garlan et al., 2003] Garlan, D., Khersonsky, S., and Kim, J. (2003). Model checking publish-subscribe systems. In *Proceedings of the 10th International SPIN Workshop on Model Checking of Software (SPIN03)*, Portland, Oregon.
- [Gaver et al., 1999] Gaver, W., Dunne, T., and Pacenti, E. (1999). Design: cultural probes. *ACM Interactions*, 6(1):21–29.
- [Gray and Salzman, 1998] Gray, W. and Salzman, M. (1998). Damaged merchandise? a review of experiments that compare usability evaluation methods. *Human Computer Interaction*, 13(3):203–261.
- [Halnass and Redstrom, 2002] Halnass, L. and Redstrom, J. (2002). From use to presence: On the expressions and aesthetics of everyday computational things. *ACM Transactions on Computer-Human Interaction*, 9(2):106–124.
- [Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274.
- [IST, 2004] IST, Advisory, G. (2004). Experience and application research: Involving users in the development of ambient intelligence. Technical report, ISTAG Working Group. Final Report.
- [Lewis et al., 1990] Lewis, C., Polson, P., Wharton, C., and Rieman, J. (1990). Testing a walk-through methodology for theory based design of walk-up-and-use interfaces. In Chew and Whiteside, editors, *ACM-CHI 90*, pages 235–242. Addison-Wesley.
- [Loer, 2003] Loer, K. (2003). *Model-based Automated Analysis for Dependable Interactive Systems*. PhD thesis, Department of Computer Science, University of York, UK.
- [Loer and Harrison, 2004] Loer, K. and Harrison, M. (2004). Analysing and modelling context in mobile systems to support design. Technical Report CS-TR-876, School of Computing Science, University of Newcastle upon Tyne.
- [Loer and Harrison, 2005] Loer, K. and Harrison, M. (2005). Analysing user confusion in context aware mobile applications. In Constabile, M. and Paternò, F., editors, *INTERACT 2005*, number 3585, pages 184–197. Springer Lecture Notes in Computer Science.

- [Loer and Harrison, 2006] Loer, K. and Harrison, M. (2006). An integrated framework for the analysis of dependable interactive systems (ifadis): its tool support and evaluation. *Automated Software Engineering*. in press.
- [Loer et al., 2004] Loer, K., Hildebrandt, M., and Harrison, M. (2004). Analysing dynamic function scheduling decisions. In *Human Error, Safety and Systems Development*, pages 45–60. Kluwer Academic.
- [McCarthy and Wright, 2004] McCarthy, J. and Wright, P. (2004). *Technology as Experience*. MIT Press.
- [McMillan, 1993] McMillan, K. (1993). *Symbolic model checking*. Kluwer.
- [Monk et al., 1993] Monk, A., Wright, P., Haber, J., and Davenport, L. (1993). *Improving your human-computer interface: a practical technique*. Prentice-Hall.
- [Nielsen, 1992] Nielsen, J. (1992). Finding usability problems through heuristic evaluation. In *Proc. of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 249–256, New York. ACM.
- [Nilsson et al., 2000] Nilsson, J., Sokoler, T., Binder, T., and Wetcke, N. (2000). Beyond the control room: mobile devices for spatially distributed interaction on industrial process plants. In Thomas, P. and Gellersen, H.-W., editors, *Handheld and Ubiquitous Computing, HUC'2000*, number 1927 in Springer Lecture Notes in Computer Science, pages 30–45. Springer.
- [Pocock et al., 2001] Pocock, S., Harrison, M., Wright, P., and Johnson, P. (2001). THEA: A technique for human error assessment early in design. In Hirose, M., editor, *Human-Computer Interaction INTERACT'01 IFIP TC.13 International Conference on human computer interaction*, pages 247–254. IOS Press.
- [Rosson and Carroll, 2002] Rosson, M. and Carroll, J. (2002). *Usability Engineering: scenario-based development of human computer interaction*. Morgan Kaufman.
- [ten Beek et al., 2006] ten Beek, M., Massink, M., and Latella, D. (2006). Towards model checking stochastic aspects of the thinkteam user interface. In Harrison, M. and Gilroy, S., editors, *Proceedings 12th International Workshop on the Design, Specification and Verification of Interactive Systems*. Springer Lecture Notes in Computer Science. in press.

Supporting Usability Evaluation of Multimodal Man-Machine Interfaces for Space Ground Segment Applications Using Petri net Based Formal Specification

Philippe Palanque*

University Paul Sabatier, Toulouse, 31062, France

Regina Bernhaupt†

Universität Salzburg, Salzburg, 5020, Austria

David Navarre‡

University Toulouse 1, Toulouse, 31042, France

Mourad Ould§

CNES, Toulouse, 31041, France

and

Marco Winckler**

University Paul Sabatier, Toulouse, 31042, France

This paper describes the issues raised by the evaluation of multimodal interfaces in the field of command and control workstations. Design, specification, verification and certification issues for such Man-Machine Interfaces (MMIs) have been already identified as critical activities. This paper focuses on the issues raised by evaluation of their usability evaluation. We first present a formalism (Interactive Cooperative Objects) and its related case tool (PetShop) for the specification of such MMIs and then show how the models built can support the usability evaluation phase. As a case study we present a multimodal interaction for 3D navigation in a 3D satellite model.

I. Introduction

The importance of the Man-machine Interface (MMI) part of Space Ground Segment Information Treatment applications is significant and increasing. The same holds for costs of these MMIs throughout the various phases of their life cycle, namely design, development, operation and maintenance.

Current research work in the field of Human Computer Interaction promotes the development of new interaction and visualization techniques in order to increase the bandwidth between the users and the systems they are interacting with. Such an increase in bandwidth can have a significant impact on efficiency. For instance the number of commands triggered by the users within a given amount of time and the error rate, typically the number of slips or mistakes⁴⁰ made by the users, are influenced by the user interface.

On the interaction technique side, these new technologies promote the use of multimodal interfaces allowing users to interact with the system by entering data or commands using a combination of several input devices. In addition to "traditional" devices such as keyboard and mouse other "new" devices are available for the designers. Such devices include tactile screens, voice recognition systems, speech synthesisers, haptic devices (possibly providing force feedback) and eye tracking (allowing visual pointing).

* Professor, LIIHS-IRIT, 118, Route de Narbonne, 31062 Toulouse cedex 4.

† Dr. ICT&S-Center, Universität Salzburg Sigmund-Haffner-Gasse 18, 5020 Salzburg.

‡ Dr. Place Anatole France, 31042 Toulouse cedex.

§ CNES (Centre National d'Etudes Spatiales), 18 avenue Edouard Belin, 31041 Toulouse cedex 4.

** Dr. Place Anatole France, 31042 Toulouse cedex.

The research results presented in this paper have been produced within a research project that targeted three main objectives:

- study the potential contribution of these new MMI technologies for our Space Ground Segment IT software,
- evaluate their adequacy with our needs in order to anticipate the design problems of the future MMIs and at the same time to assess our space systems,
- Develop interactive applications with the same quality as non interactive applications i.e. through a rigorous development process.

One of the key issues of the research carried out in the project is to find efficient ways of bringing together two separated (and often opposite) issues such as usability and reliability. Indeed, the continuously increasing complexity of the information manipulated by safety critical interactive systems calls for new interaction techniques increasing the bandwidth between the system and the user. Multimodal interaction techniques are considered as a promising way of tackling this problem. However, the lack of engineering techniques and processes for such systems makes them hard to design and to build and thus jeopardizes their actual exploitation in the area of safety critical applications. Space Ground Segment IT software is one the application domains where such failures can have a catastrophic impact on the equipment or data under manipulation.

In a previous paper³⁶ we presented the challenges provided by multimodal and 3D user interfaces as far as software design, specification and verification are concerned. We proposed a new formalism dedicated to the formal specification of such interfaces and presented a software CASE (Computer Aided Software Engineering) tool dedicated to this formalism. This tool called PetShop (for Petri nets workshop) allows software developers to edit and modify the models of the multimodal interface. This tool, a tutorial and a set of examples are available on our research group web site: <http://lihs.irit.fr/petshop>

The current paper is dedicated to the last phase of the research project. After the phase of defining notations and tools for the specification of multimodal interfaces, we are presenting how these components can support the usability evaluation phase. Indeed, the fact that both the interaction technique and the entire application have been formally specified makes it possible to exploit that information (usually not available in a “classical” development process of interactive applications).

The paper is structured as follows. The next section (section 2) presents the state of the art in the field of usability evaluation methods for multimodal interfaces. This section first presents a brief introduction about multimodal interaction and multimodal interfaces and then compares the current practice in multimodal interfaces usability evaluation. Section 3 informally presents the Interactive Cooperative Objects formal description technique. It also describes the specificities of multimodal interfaces and their impact on the expressive power and verification techniques. Section 4 introduces the case study used for providing a concrete example of multimodal interaction techniques and for showing how the ICO formalism is applied. The case study is used here, as a proof of concept and does not correspond to any Space Ground Segment application currently deployed. The last section (section 5) details through the explicit representation of user goals, tasks and interaction scenarios how usability evaluation could be conducted for these kinds of systems. It also shows how the formal specification technique can provide useful precise information for supporting this task.

II. Usability evaluation of multimodal interfaces

A. Introduction to interactive systems and multimodality

By definition, multimodal interaction techniques make it possible for the users to interact with the application using several modalities. A modality is defined as a couple made up of a device (input or output) and an interaction language. Thus multimodality can take place in two different ways:

- input multimodality that involves the use of several input devices such as a mouse, keyboard, voice recognition system, gaze recognition,...;
- output modality that involves the use of several output devices such as screen, spatialized sound, 3D visualisation systems, ...



Figure 1. Two handed interaction on a tablet PC (extracted from⁴⁵)

Today, multimodal interaction techniques are used in almost all areas ranging from business software to embedded systems such as cockpits of military aircrafts⁷.

In this project we focus on input multimodality supporting the use of speech input and input by two mice. The successful use of two handed interaction has been shown in⁹. Using a toolglass and magic lenses, the user is able to select a color at the same time as choosing an object. A similar two handed interaction for coloring objects has been shown in⁴⁵ for pen and touch input (see Figure 1).

A previous study³⁷ has shown that, in the field of safety critical systems, multimodal interaction presents several advantages.

- Multimodality increases reliability of the interaction. Indeed, it permits to drastically decrease critical error (between 35% and 50%) during interaction. This advantage, on its own, can justify the use of multimodality when interacting with safety critical systems.
- It increases efficiency of the interaction, in particular in the field of spatial commands (multimodal interaction is 20% more rapid than classical interaction to specify geometric and localization information).
- Users predominantly prefer interacting in a multimodal way, probably because it allows more flexibility in interaction thus taking into account users' variability.
- Multimodality allows increasing naturalness and flexibility of interaction so that the learning period is shorter.

However, assessing quantitatively and in a predictive way both efficiency and usability of multimodal interactive systems is still considered as a difficult problem by the research community in the field of human-computer interaction. To structure the issues, a set of multimodal properties, called the CARE properties, have been identified in¹⁷. According to this study, input and output modalities can be combined in four ways: complementarily, assignment, redundancy and equivalence. These properties can be used both at design time when designers have to define how multimodal interaction will take place and also at exploitation time when the users will actually use the system and select how they will interact with the system. For instance if the designer provides two equivalent modalities for triggering a command, the user will be able to choose any of them while interacting with the system. Assignment of a given modality to a given command requires, on the other side, the user to use only that modality for triggering this command.

As CARE properties are more oriented towards design and use of modalities, they are of little help as far as the precise specification or the construction of the system is concerned. The other classification¹⁷ proposed a system's view on applications featuring multimodal interactions. Indeed, one of the key elements of multimodal interfaces is related to the fusion of information provided by several devices used in a concurrent way by the user.

This classification (see Table 1) is structured according to 2 criteria:

1. Use of modalities: this criterion indicates if two modalities can be used in parallel (i.e. at the "same" time) or in sequence (i.e. one after the other),
2. Interpretation: this criterion indicates if pieces of information coming from two different modalities are fused to define a new command. This is the case, for instance, when the user triggers the command using voice recognition and provides the parameter for this command using the mouse like in saying the word "delete" and clicking simultaneously on a graphical object to be deleted.

		Use of modalities	
		Sequential	Parallel
Interpre- tation	Combined	Alternating.	Synergistic
	Independent	Exclusive	Concurrent

Table 1. Usage and interpretation of modalities

The table shows that there can be four kinds of multimodal interaction both addressing input and output:

- **Exclusive:** is the poorer kind of multimodality. Modalities can only be used in a sequential way and information provided by two different modalities are not fused.
- **Concurrent:** Modalities can be used both in sequence and in parallel, but information provided by two different modalities are not fused for triggering a command. So modalities have to be used for different tasks and in a non correlated way.
- **Alternating:** Modalities can only be used in a sequential way and information provided by the different modalities is fused. Fusion/fission mechanisms can be applied to trigger specific commands if the information content is compatible.
- **Synergistic:** it is the more complex and powerful kind of multimodality. Modalities can be used both in sequence and in parallel and information provided by two different modalities can be fused.

B. Usability Evaluation methods for Multimodal Interfaces

Multimodal User Interfaces (MUI) are often referred to be more natural to users than “uni-modal” user interfaces (e.g. graphical UI) because they allow making better use of users’ communication channels. This assumption is supported by studies on two-handed multimodal user interfaces which have shown that using two pointing devices in a normal graphical user interface has been found to be more efficient and understandable than the basic mouse and keyboard interfaces as described in (Ref. 13, 26, 46). However, multimodal interaction only benefits when the design takes into account the human abilities and the appropriate selection of communication channels. Several studies have revealed that when these interfaces are designed poorly, they are neither better understood nor more efficient (Ref. 19, 26).

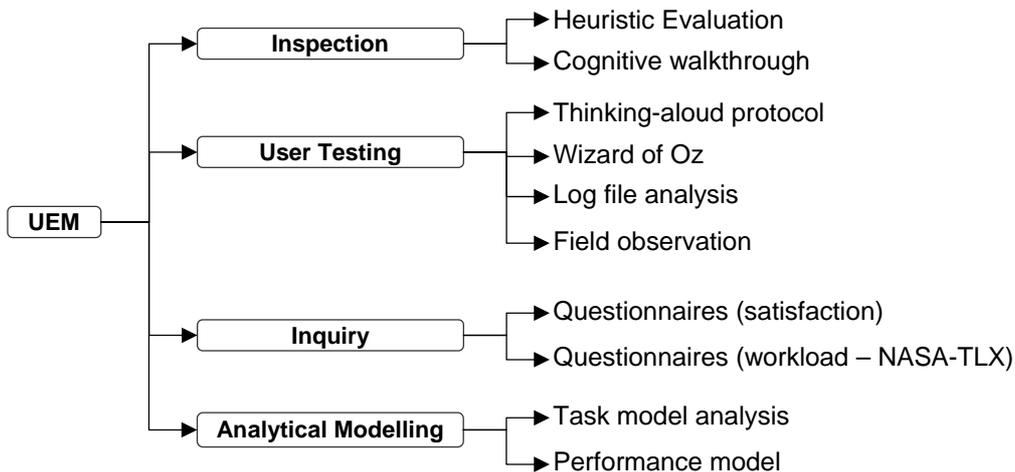


Figure 2. An overview of evaluation methods for multimodal user interfaces.

The combinations of input and output devices and interaction modalities have opened a complete new world of experience for interactive systems and it poses the question on how to accurately evaluate the usability of multimodal user interfaces. The results of existing empirical studies of such as multimodal applications revealed a very intricate problems concerning the assessment of multimodal technology where individual user preferences for modality¹⁸, context of use and kind of activity supported by the system (task-oriented versus non-task oriented)²⁶,

use of specific devices^{22,24} and interaction technique³² (e.g. pointing x clicking), play major role to determine the usability.

There have been attempts to adapt traditional usability evaluation methods for use in multimodal systems, and a few notable efforts to develop structured usability evaluation methods for multimodal applications. Regarding the current practice of usability evaluation of multimodal systems, we may distinguish in Figure 2 four main approaches: a) theoretical frameworks based on inspection methods, b) empirical investigation based on user testing, c) user inquiry, and d) analytical modeling.

Despite the great importance of inspection methods such as Heuristic Evaluation³³ and cognitive walkthrough²⁸ for usability evaluation of user interfaces, they have been found less useful to assess multimodal user interfaces.

The great majority of studies still employ some kind of user testing where user activity is measured while performing some pre-defined tasks. Log file analysis³⁹ and the think-aloud protocol is frequently employed in both laboratory and in field observation²⁵ with advanced prototypes. Nevertheless, mockups and early prototypes have also been testing using the method Wizard of Oz²⁷ technique. User testing seems to be a preferred strategy for evaluation of many studies of multimodal user interfaces because it allows the investigation of how users adopt and interact with multimodal technology.

Questionnaires have been extensively employed to obtain qualitative feedback from users (e.g. satisfaction, perceived utility of the system, user preferences for modality) as well to assess cognitive workload⁴² (especially using the NASA-TLX method). Questionnaires have quite often been used in combination with user testing techniques²⁵.

More recently, two main approaches have been developed to support analytical modeling which intends to predict usability of multimodal user interfaces. On one hand simulation and model-based checking of task specifications are used to predict usability problems such as unreachable states of the systems or conflicting events required for fusion. (Ref. 39) propose to combine task specification based on ConcurTaskTree (CTT) with multiple data sources (e.g. eye-tracking data, video records) in order to better understand the user interaction and the task models used to support the development process of multimodal user interfaces. On the other hand, analytical modeling based on fitness functions³⁸ and Fuzzy Logical Model of Perception²⁹ (FLMP) that belongs to the category of predictive analytical modeling try to mathematically explain how users interact with the system. These approaches highlight how humans benefit from multiple sources of information from multiple modalities but they cannot be used alone for usability assessment.

III. ICO a formal description technique for safety critical interactive systems

Design, specification and verification of interactive systems is very complex and classical techniques and tools in the field of software engineering do not provide adequate support for these kinds of software systems. Complexity increases when more sophisticated interaction techniques (such as multimodal interaction) are made available to the users. We believe that the use of an adequate formal description technique can provide support for a more systematic development of multimodal interactive systems. Indeed, formal description techniques allow for describing a system in a complete and non-ambiguous way thus allowing an easier understanding of problems between the various persons participating in the development process. Besides, formal description techniques allow designers to reason about the models by using analysis techniques. Classical results can be the detection of a deadlock or presence or absence of a terminating state. As stated above, a set of properties for multimodal systems have been identified¹⁷ but their verification over an existing multimodal system is usually very difficult to achieve. For instance it is very difficult to guarantee that two modalities are redundant whatever state the system is in.

C. Related Work on Engineering Multimodal Applications

This paper exploits a formal description technique that we have defined. This proposal builds upon previous work we have done in the field of formal description techniques for interactive systems and is an answer to several requests from industry to provide software engineers with software engineering techniques for multimodal systems.

Work in the field of multimodality can be sorted into five main categories. Of course, the aim of this categorization is not to be exhaustive but to propose an organization of previous work in this field.

- **Understanding multimodal systems:** (Ref. 16) presents a typology of multimodal systems while (Ref. 17) deals with properties of multimodal systems.
- **Software construction of multimodal systems:** (Ref. 8) and (Ref. 14) propose toolkits for the construction of multimodal systems, and (Ref. 34) proposes a generic software architecture for multimodal systems.
- **Analysis and use of novel modalities:** (Ref. 11) presents the first use of voice and gesture as combined modalities. (Ref. 12) introduces two handed interaction (Ref. 10) introduces the use of two handed

interaction for virtual reality applications and (Ref. 44) presents Jeanie, a multimodal application, to test the use of eye tracking and lips movements recognition.

- **Multimodal systems description:** (Ref. 15) presents QuickSet a cooperative interface using both voice and gesture, while (Ref. 34) presents a Multimodal Air Traffic Information System (MATIS) using both voice and direct manipulation interaction. Similarly, (Ref. 9) presents a drawing system featuring two handed interaction through a trackball and a mouse.
- **Multimodal systems modeling:** (Ref 1) exploits high-level Petri nets for modeling two handed interaction (a mouse and a trackball) and (Ref 18 and Ref 45) use finite state automats for modeling two handed interaction.

D. Informal Description of ICOs

The aim of this section is to recall the main features of the ICO (Interactive Cooperative Objects) formalism that we have proposed for the formal description of interactive system. The formalism will be used for the case studies and performance evaluation in the next sections. We encourage the interested reader to look at (Ref. 4) and (Ref. 43) for a complete presentation of this formal description technique.

The Interactive Cooperative Objects (ICO) formalism is a formal description technique dedicated to the specification of interactive systems (Ref. 5). It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance, client/server relationship) to describe the structural or static aspects of systems, and uses high-level Petri nets (Ref. 22) to describe their dynamic or behavioral aspects.

ICOs are dedicated to the modeling and the implementation of event-driven interfaces, using several communicating objects to model the system, where both behavior of objects and communication protocol between objects are described by Petri nets. The formalism made up of both the description technique for the communicating objects and the communication protocol is called the Cooperative Objects formalism (CO and its extension to CORBA COCE (Ref. 43)).

In the ICO formalism, an object is an entity featuring four components: a cooperative object with user services, a presentation part, and two functions (the activation function and the rendering function) that make the link between the cooperative object and the presentation part.

Cooperative Object (CO) part: a cooperative object models the behavior of an ICO. It states how the object reacts to external stimuli according to its inner state. This behavior, called the Object Control Structure (ObCS) is described by means of high-level Petri net. A CO offers two kinds of services to its environment. The first one, described with CORBA-IDL³⁵, concerns the services (in the programming language terminology) offered to other objects in the environment. The second one, called user services, provides a description of the elementary actions offered to a user, but for which availability depends on the internal state of the cooperative object.

Presentation part: the Presentation of an object states its external appearance. This Presentation is a structured set of widgets organized in a set of windows. Each widget may be a way to interact with the interactive system (user-towards-system interaction) and/or a way to display information from this interactive system (system-towards-user interaction).

Activation function: the user-towards-system interaction (inputs) only takes place through widgets. Each user action on a widget may trigger one of the ICO's user services. The relation between user services and widgets is fully stated by the activation function that associates to each couple (widget, user action) the user service to be triggered.

Rendering function: the system-towards-user interaction (outputs) aims at presenting to the user the state changes that occurs in the system. The rendering function maintains the consistency between the internal state of the system and its external appearance by reflecting system states changes.

ICOs are used to provide a formal description of the dynamic behavior of an interactive application. An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays information relevant to the user).

An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented (Ref. 6). The specification can also be validated using analysis and proof tools developed within the Petri nets community and extended in order to take into account the specificities of the Petri net dialect used in the ICO formal description technique. This formal specification technique has already been applied in the field of Air Traffic Control interactive applications. A case study on this field can be found in (Ref. 30).

IV. Multimodal interaction on a 3D satellite model

This section presents the exploitation of the formalism presented in previous sections to a case study in the field of space applications. Even though the application is not one currently used by “real” users it has been designed in order to cover a wide range of issues spanning from formal specification of interactive behaviors to usability evaluation.

E. Informal presentation of the case study

This application provides multimodal interaction techniques to a user moving the point of view (we will later call this “navigating”) in a 3D model of a satellite. This navigation can be done either by rotating the 3D model of the satellite directly using the mouse on the 3D image or using the two control panels presented in Figure 3.

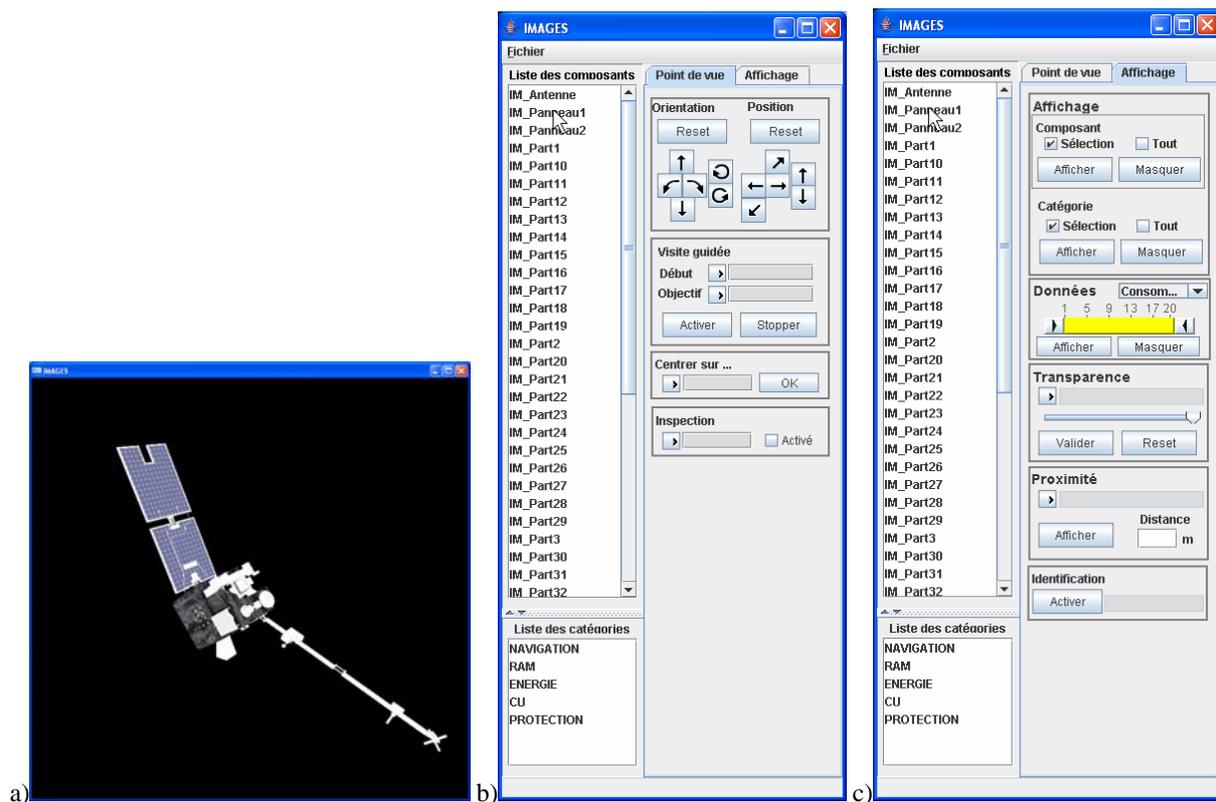


Figure 3. The 3D representation of Demeter satellite (a) and its two control panels (b and c)

The control panel (b) entitled “point de vue” allows the user to manipulate the current position of the point of view of the 3D image using the set of buttons in the top right hand side of Figure 3b. The set of buttons in the “orientation” section allows rotating the satellite image in any direction. The two list-boxes on the left hand side present respectively the list of components of the satellite and the list of categories the components belong to. We do not present the other parts of the user interfaces as they are beyond the scope of this paper.

At the beginning the satellite appears as presented in Figure 3a). The main task given to the user of this application is to locate one or several components in the satellite. This task is not easy to perform as components are nested and might not be visible. Thus the user interface offers the possibility to make the set of components selected from the list partly or fully transparent. This transparency is set by means of the Transparence slider on the right hand side of Figure 3c). The goal of the user is to locate components that can be of two types: overheating and over-consuming. The selection of the range of temperature of interest and the range of consumption can be done using the range slider in the section “données” in the right hand side of Figure 3c). Figure 4 presents a snapshot of the satellite 3D model including the temperature of the visible components.

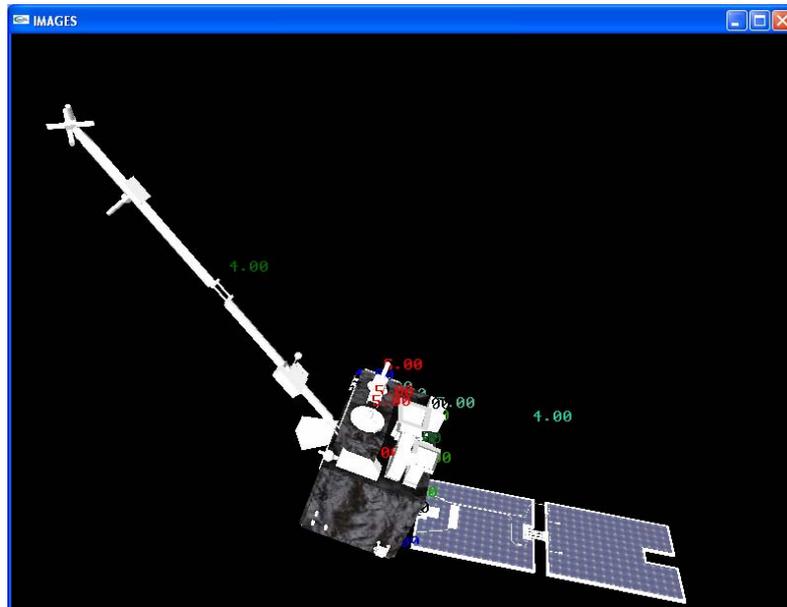


Figure 4. 3D satellite model displaying the temperature of the visible components

In this application multimodal interaction takes place both while using the button pairs, changing the point of view of the 3D model, and while interacting with the range slider for selecting the temperature and the consumption.

Due to space constraints we only present here multimodal interaction on the button pair. The interested reader can see the formal specification of a similar multimodal range slider component in (Ref 21).

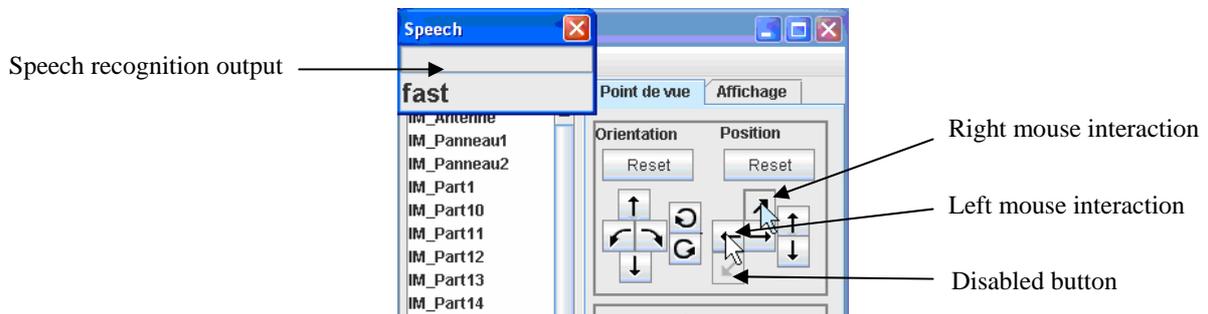


Figure 5. One of the multimodal interactions in the application

Figure 5 shows the multimodal interaction in action. In this figure the user is currently using 3 input devices at a time: 2 mice and 1 speech recognition system. The speech recognition systems allows only for entering 2 different words: “fast” and “slow”. The interaction takes place in the following way: at any time the user can use any of the mice to press on the buttons that change the point of view. In Figure 5, the button that moves the satellite image backwards (with the additional label right mouse interaction on Figure 5) has been pressed using the right mouse. Simultaneously the left mouse has been positioned on the button moving the satellite image to the left. At that time the image has already started to move backwards and as soon as the other button will be pressed the image will be moving both backwards and to the left. The user is also able to increase or decrease the movement speed by uttering the words “fast” and “slow”. In Figure 5 the word “fast” has been pronounced and recognized by the speech recognition system and is thus (as shown on the left-hand side of Figure 5). This action will reduce the time between two movements of the image. Indeed, the image is not moved according to the number of clicks on the buttons but according to the time the buttons are kept pressed by the user.

Describing such interaction techniques in a complete and unambiguous way is one of the main issues to be solved while specifying and developing multimodal interactive systems. The next section presents how the ICO formalism is able to deal with these issues. Additionally it will show that the description above is incomplete and does not

address at an adequate level of detail both timed and concurrent behavior at least when it comes to implementation issues.

F. ICO modeling of the case study

This section is devoted to the formal modeling of the multimodal interactive application presented in section E. In this multimodal application there is no fusion engine per se, the two mice are handled independently and the speech interaction affects movement speed whatever interaction is performed with the mice.

The modeling is structured as represented in Figure 6. The right hand side of the figure shows the user interacting with the input devices. As stated before three input devices are available. In order to configure this set of input devices we use a dedicated notation called Icon¹⁴. A more readable model of this configuration is represented in Figure 7.

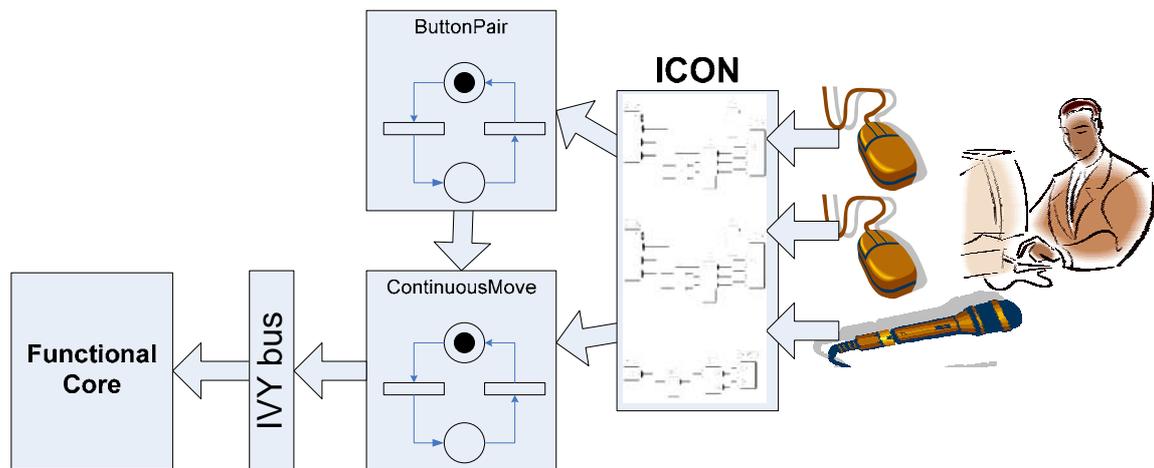


Figure 6. Software architecture of the multimodal interactive application

The left hand side of Figure 7 represents the 3 input devices connected to software components. These components are represented as graphical bricks and connectors model the data flow between these bricks. For instance it defines that interaction with the mice will take place using the left button (but1 in the usbMouse brick) and that the alternate button for the speech recognition system is the space bar (Space label in the keyboard brick connected to the speechCmd brick). The right hand side of this figure represents contact points with the other models of the application. As input configurations are not central to the scientific contribution of this paper we do not present in more detail how this modeling works. More information about the system supporting the edition and execution of models, the behavior of a model and the connections to other models can be found in (Ref. 31). Similarly, functional core and communication protocol between the functional core and the interaction models are not presented.

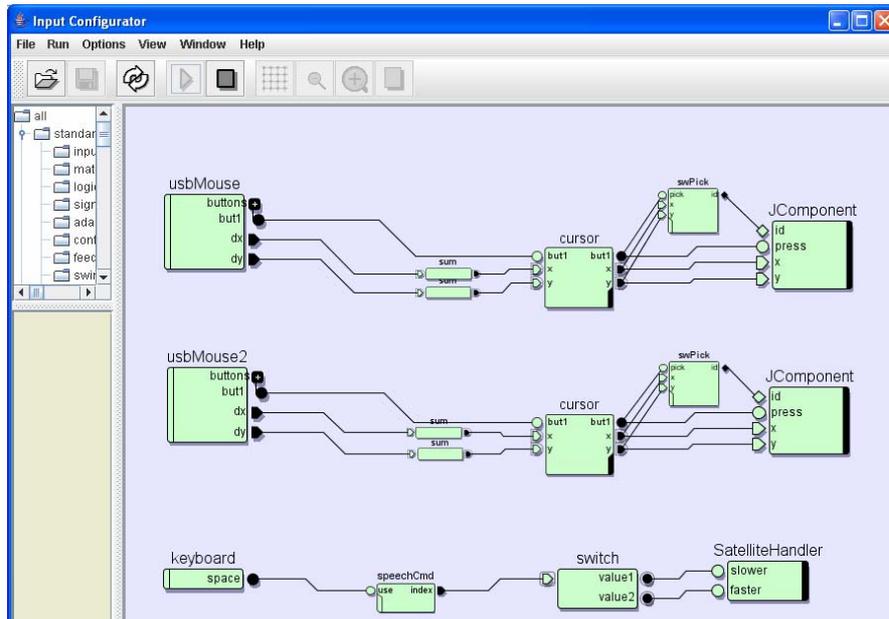


Figure 7. Input configuration using ICon¹⁴

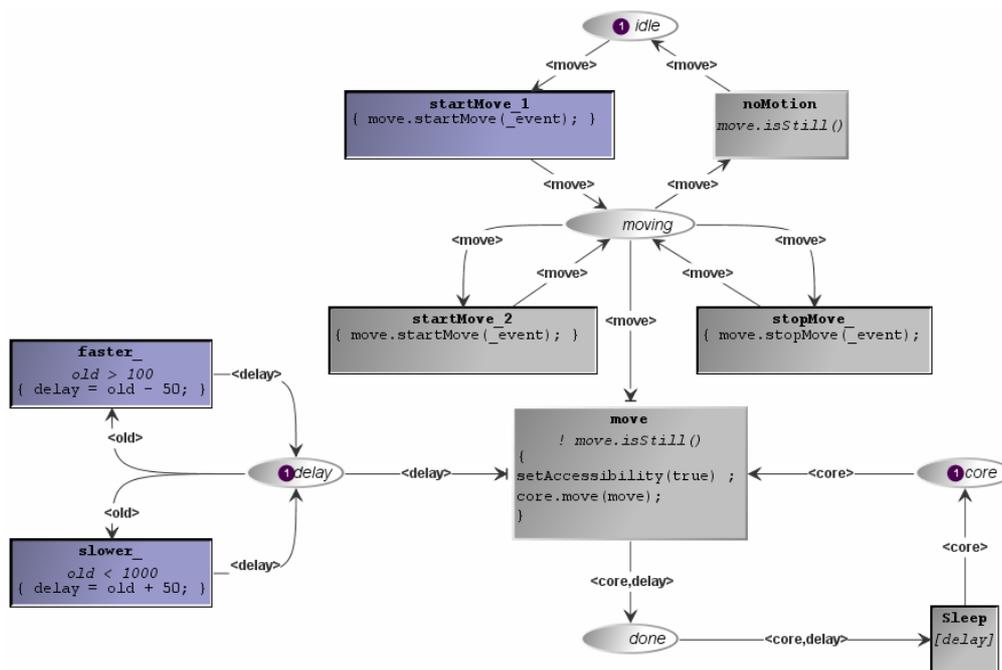


Figure 8. Model of the temporal evolution of movements driven by speech (continuous move in Figure 6)

The ICO model in Figure 8 represents the complete and unambiguous temporal behavior of the speech-based interaction technique as well as how speech commands impact the temporal evolution of the graphical representation of the 3D image of the satellite. Darker transitions are available according to the current marking of the models. Taking into account the current marking of the model of Figure 8 (one token in each place *delay*, *Idle* and *core*) only transitions *startMove_1*, *faster_* and *slower_* are available. These transitions describe the multimodal interaction technique available i.e. how each input device can be used to trigger actions on the system. Transitions *faster_* and *slower_* are triggered when the user utters one of the two speech commands fast and slow. In the initial state these are available and will remain available until the upper limit and the lower limit are reached (respectively $delay > 1000$ for transition *slower_* and $delay < 100$ for transition *faster_*).

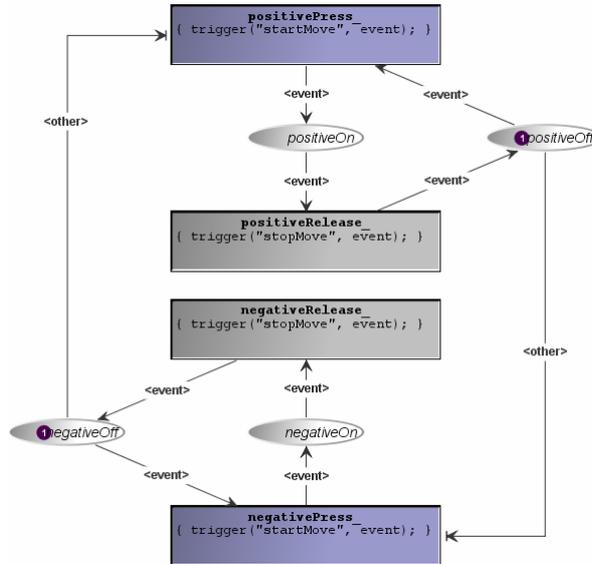


Figure 9. Mutual exclusion of the pair of buttons for changing the point of view (button pair in **Figure 6**)

Figure 9 presents another model of the application, responsible for describing the behavior of each button pair. By button pair we mean the buttons that are antinomic, namely (up, down), (left, right) and (backwards, forward). These 3 button pairs are represented on the right hand side of Figure 5. To model this antinomy the ICO description represents the fact that the user can press either the positive or negative button. Once pressed, these buttons can be released when the left button on the input device is released (as represented in the ICon model of Figure 7).

G. Low level interaction constraints

Models of Figure 8 and Figure 9 integrate some constraints for the user. These constraints can result from the design process and design choices (the 3D image will not move faster than one modification every 100 milliseconds) or physical constraints (one button on a mouse can only be either pressed or released).

All of these constraints are made explicit in the models and thus provide a unique source of information about the actual precise behavior of the interaction technique. Thanks to the expressive power of the underlying Petri net formalism used in ICO, concurrent behavior can be described together with quantitative temporal evolution (the image will change every 100 milliseconds). Indeed, Petri nets is the only formalism able to express concurrency and both quantitative and quantitative temporal evolutions.

For example the ICO model makes explicit the speech-driven temporal evolution of the model. The value of the delay between two images is stored in a variable called Delay. This variable is then used as the timer in the transition Sleep (bottom right hand side of Figure 8). This construct is defined in Generalized Stochastic Petri nets³ and behaves as follow. According to the buttons pressed using the mice, a token containing this information will be set in place *moving*. While the buttons are pressed the transition **move** is fired performing the calculation of the 3D image and rendering it on the screen. This will move the token from place *core* to place *done* making the transition **move** unavailable (there is no token in one of its input place (the place *core*)). Transition **Sleep** will become available (one token in each input place (only place *done*)) but will not fire as it is a timed transition. Indeed, the transition will wait until the amount of time delay (the variable delay is storing a number of milliseconds) has elapsed before firing. When this amount of time has elapsed the transition will fire removing the token from place *done* and setting a new one in place *core*. After this loop, transition **move** will become available again and thus be ready to render a new 3D image of the satellite. As transitions **startMove_2** and **stopMove** remain available during the loop it makes explicit the fact that users can press or release any button on the user interface at any time.

V. Model-Based Usability Evaluation

H. Basic Principles of Model-Based Evaluation

As presented in section B, to ensure usability of these kinds of applications, various methods from the field of human-computer interaction (HCI) can be applied. Even though any kind of usability evaluation method can help in this process we will focus on applying usability tests.

A typical usability test is performed in a laboratory (sometimes in the field), where users are asked to perform selected tasks. The users are observed by cameras, and they might be asked to talk aloud (also called elicitation activity) while performing the task. A usability test typically begins with asking the users a pre-questionnaire related to the domain of the software (use of other related systems, experience with multimodal-interfaces, hours of training ...). Some tasks are then performed to ensure that the user is able to use the system. Testing multimodal interactions usually requires an additional activity corresponding to the presentation of the key input modalities to the user. The user then performs the tasks. Tasks have to be completed within a given time. If the user cannot solve the task within this time, the experimenter (leader of the usability test) helps the user by giving hints or providing the solution. The number of successful completions and the completion time are recorded. Tasks not solved indicate usability problems, leading to further detailed investigations of the problems. For an example of a usability test recording see Figure 10.



Figure 10. Example of usability test in action

When complex interaction techniques are considered (as in the current application) the presentation of the application to be tested with the user also requires a description of the actual interaction technique. This description goes beyond the typical high-level (task-based) scenarios promoted by usability testing methods. The goal of a usability test is to improve the interface by finding out major usability problems within the interface. While a common practice is to use the most frequently performed tasks (based on the task analysis), in the field of safety critical systems, it is important to cover all (or most of) the possible interactions that the user might be involved in. The explicit description, in the formal models, of the interaction techniques makes it possible to identify not only the “minimum” number of scenarios to be tested but also to select more rationally the tasks that are to be focused on.

When testing multimodal interfaces, this constraint reaches a higher level of complexity due to the significant number of possible combinations of input modalities and also due to the fact that fusion engines usually involve quantitative temporal evolution as presented in section F. In order to test all (or most) of these combinations it is required to provide usability tests scenarios at a much lower level of description than what is usually done with more

classical systems. Indeed, as for walk-up and use systems, the interaction technique must be natural enough for the user to be able to discover it while interacting with the system.

Even though we need to address this issue of low level scenarios it is also important to notice that usability testing is very different from software testing. The objective here is to test the usability of the interaction technique and not its robustness or default-freeness like in classical software testing. The issue of reliability testing of multimodal interactive systems is also very important but it is beyond the scope of this paper. Formal methods can help to specify the “real” number of low-level interaction scenarios and thereby inform selection of tasks more appropriately.

It is important to note that we are not claiming that current practices in the field of usability evaluation must involve model-based usability evaluation. Our claim is that in the field of safety critical interactive systems and more specifically when multimodal interaction techniques are considered, model-based approaches can support specific activities (like low-level tests scenarios and tasks identification) that could be otherwise overlooked or not systematically considered.

The next section presents examples of low-level interaction scenario descriptions for the case study.

I. Example of Model-Based Evaluation on the Case Study

The description of the temporal evolution presented in section G shows how complex low-level multimodal interaction can be. When it comes to testing the usability of such behavior it is required first to provide a detailed description of the behavior to the evaluators but also to make it possible to modify such behavior if the results of the usability testing require to do so. Some of the modifications in the ICO model of Figure 8 are trivial:

- Changing the value of increase and decrease of time when speech commands are issued: this can be done by changing the line $delay=old+50$ in the transition **slower_** for instance to another amount of increase
- Changing the maximum speed of 3D image rendering: this can be done by changing the precondition in transition **slower_** or **faster_** to another value than 1000 (maximum) and 100 (minimum).

Other complex behaviors relative to qualitative temporal behaviors can also be represented and thus exploited during usability tests. For instance as modeled in Figure 8 all the input modalities are available all the time but another design choice could have been to allow only to use a maximum of 2 input modalities at a time. In such case these limitation should have been presented with precise details to the user before executing the evaluation scenarios. Similarly, some scenarios could have been selected with the explicit purpose of evaluating comfort and cognitive workload induced by this kind of reduction of the interaction space.

This notion of low-level interaction technique can have a significant impact on the results and thus the interpretation of usability tests results. We are currently in the phase of performing such model-based evaluation on a real ground segment information treatment system to assess the impact of multimodal interaction techniques on the ease of use and performance. The goal is also to assess impact of model-based evaluation with respect to more classical usability evaluation technique for multimodal systems (as the ones presented in section B).

VI. Conclusion

This paper has presented the use of a formal description technique for describing multimodal interactive applications. Beyond that, we have shown that this formal description technique is also adequate for interaction techniques and low level interactive components. One of the advantages of using the ICO formal description technique is that it provides additional benefits with respect to other notations such as Statecharts⁴¹. Thanks to its Petri nets basis, the ICO notations makes it possible to model behavior featuring an infinite number of states (as states are modeled by a distribution of tokens in the places of the Petri nets). Another advantage of ICOs is that they allow designers to use verification techniques at design time as has been presented in (Ref. 30). These verification techniques are of great help for certification purposes. Beyond these software engineering benefits, we have also shown that this model-based approach can also support the usability evaluation activities that are usually considered externally from the actual development process. This specific contribution provides a first step for integrating, in a same development framework, requirements coming both reliability and usability communities.

Acknowledgments

This work is partly funded CNES (National Center of Spatial Studies in France), DGA (French Army Research Dept.) under contract INTUITION #00.70.624.00.470.75.96, EU via the ADVISES Research Training Network RTN2-2001-00053 and Network of Excellence ResIST (www.resist-noe.org).

References

- ¹ Accot J. Chatty S. and Palanque P. A Formal Description of Low Level Interaction and its Application to Multimodal Interactive Systems, 3rd EUROGRAPHICS workshop on Design, Specification and Verification of Interactive systems, Springer-Verlag, pp. 92-104, 1996.
- ² Accot, J., Zhai, S. Performance evaluation of input devices in trajectory-based tasks: an application of the steering law. Proceedings of the SIGCHI conference on Human factors in computing systems CHI'99, p. 466-472. ACM Press.
- ³ Ajmone Marsan M.; Balbo G.; Conte C.; Donatelli S., and Franceschinis G. "Modelling with generalized stochastic Petri nets". Wiley; 1995.
- ⁴ Bastide R. and Palanque P. A Petri-Net Based Environment for the Design of Event-Driven Interfaces. 16th International Conference on Applications and Theory of Petri Nets, ICATPN'95, Torino, Italy. Lecture Notes in Computer Science, no. 935. Springer verlag, 1995 p.66-83.
- ⁵ Bastide R., and Palanque P. A Visual and Formal Glue Between Application and Interaction. Journal of Visual Language and Computing 10, no. 3 (1999)
- ⁶ Bastide R., Navarre D. & Palanque P. A Model-Based Tool for Interactive Prototyping of Highly Interactive Applications, *Proceedings of the ACM SIGCHI 2002 (Extended Abstracts)*, pp. 516-517, 2002.
- ⁷ Bastide R., Navarre D., Palanque P., Schyn A. & Dragicevic P. A Model-Based Approach for Real-Time Embedded Multimodal Systems in Military Aircrafts. 6th ACM International Conference on Multimodal Interfaces (ICMI'04) October 14-15, 2004 Pennsylvania State University, USA, pp. 245-258.
- ⁸ Bederson B., Meyer J. & Good L. Jazz an Extensible Zoomable User Interface Graphics Toolkit in Java, UIST'2000, ACM Symposium on User Interface Software and Technology, pp. 171-180, 2000.
- ⁹ Bier E. A., Stone M.C., Pier K., Buxton W. and Derose T. Toolglass and Magic Lenses: The see-through interface. Computer Graphics. T. Kajiya ed.1993, pp. 73-80.
- ¹⁰ Bolt R. & Herranz E. Two-Handed Gesture in Multi-Modal Natural Dialog. Proceedings of the fifth annual ACM symposium on User interface software and technology, ACM Press, California, p 7-14, 1992.
- ¹¹ Bolt R. Put That There: Voice and Gesture at the Graphics Interface, SIGGRAPH'80, p262-270, 1980.
- ¹² Buxton W. & Myers B. A Study in Two-Handed Input. Proceeding of the ACM CHI, Addison-Wesley, 1986, p 321-326
- ¹³ Buxton, W., Myers, B. A. A study in two-handed input. Human Factors in Computing Systems, CHI'86 Conference Proceedings. ACM Press, 1986, 321-326.
- ¹⁴ Chatty S. Extending a Graphical Toolkit for Two-Handed Interaction, Proceedings of the ACM symposium on User Interface Software and Technology, ACM Press, California pp. 195-204, 1994.
- ¹⁵ Cohen P., Johnston M., McGee D., Oviatt S., Pittman J., Smith I.; Chen L., & Clow J. QuickSet: multimodal interaction for distributed applications. Proceedings of the fifth ACM international conference on Multimedia; Seattle, Washington, United States . ACM Press; 1997: p 31-40.
- ¹⁶ Coutaz J. & Nigay L. A Design Space for Multimodal Systems Concurrent Processing and Data Fusion. ACM Human Factors in Computing Systems conference, INTERCHI'93, pp. 172-178, 1993.
- ¹⁷ Coutaz J., Nigay L., Salber D., Blandford A., May J. & Young R.. Four Easy Pieces for Assessing the Usability of Multimodal in Interaction: the CARE Properties. Interact' 95; Lillehammer, Norway. Chapman & Hall (IFIP); 1995: pp. 115-120.
- ¹⁸ den Os, E., de Koning, N., Jongbloed, H. and Boves. L.: Usability of a Speech-Centric Multimodal Directory Assistance Service. Proc. of the CLASS Workshop on Information Presentation and Natural Multimodal Dialogs, Verona, Italy, 2001, 65-69.
- ¹⁹ Dillon, R. F., Edey, J. D., Tombaugh, J. W. Measuring the true cost of command selection: techniques and results. Human Factors in Computing Systems, CHI'90 Conference Proceedings, ACM Press, 1990, 19-25.
- ²⁰ Dragicevic P. & Fekete J-D. Input Device Selection and Interaction Configuration with ICON. Proceedings of IHM-HCI 2001, Blandford, A.; Vanderdonckt, J.; Gray, P., (Eds.): People and Computers XV - Interaction without Frontiers, Lille, France, Springer Verlag, pp. 543-448.
- ²¹ Dragicevic P., Navarre D., Palanque P, Schyn A. & Bastide R. Very-High-Fidelity Prototyping for both Presentation and Dialogue Parts of Multimodal Interactive Systems. DSVIS/EHCI 2004 joint conference 11th workshop on Design Specification and Verification of Interactive Systems and Engineering for HCI, Germany, July 11-13, Lecture Notes in Computer Science n° 3452, Springer Verlag, 2004.
- ²² Genrich H.. Predicate/Transition Nets. High-Level Petri Nets: Theory and Application. K. Jensen and G. Rozenberg (Eds.), Springer Verlag, pp. 3-43, 1991.
- ²³ Hinckley K., Czerwinski M. and Sinclair M. Interaction and Modelling Techniques for Desktop Two-Handed Input, *Proceedings of ACM UIST*, 1998, pp. 49-58.
- ²⁴ Hinckley, K., Pausch, R., Proffitt, D., Kassel, N. F. Two-handed virtual manipulation. ACM Transactions on Computer-Human Interaction, 5 (3), 1998, 260-302.
- ²⁵ Jost, M., Haubler, J., Merdes, M., Malaka, R. Multimodal Interaction for pedestrians: an evaluation study. In Proceedings of IUI'2005, ACM Press, San Diego, USA, January 9-12.
- ²⁶ Kabbash, P., Buxton, W., Sellen, A. Two-handed input in a compound task. Human Factors in Computing System CHI'94 Conference Proceedings, ACM Press, 1994, 417-423.
- ²⁷ Klein, A., Schwank, I., Génèreux, M., Trost, H. Evaluating Multimodal Input Modes in a Wizard-of-Oz Study for the Domain of Web Search. In: Ann Blandford, Jean Vanderdonckt and Phil Gray (eds), People and Computer XV - Interaction without Frontiers: Joint Proceedings of HCI 2001 and IHM 2001, pp. 475-483. Springer: London, September.

- ²⁸ Lewis, C., Polson, P. Wharton, R. Testing a walkthrough methodology for theory-based design of walk-up-and-us interfaces. In Proceedings of CHI90, ACM Press, 1990, pp. 235-241.
- ²⁹ Massaro, D. W. A Framework for evaluating Multimodal integration by humans and a role for embodied conversational agents. In Proceedings of ICMI'2004, October 13-15 2004. Pennsylvania, USA.
- ³⁰ Navarre D., Palanque P. & Bastide R. Reconciling Safety and Usability Concerns through Formal Specification-based Development Process HCI-Aero'02 MIT, USA, 23-25 October, 2002. p. 168-175.
- ³¹ Navarre D., Palanque P., Dragicevic P. & Bastide R. An Approach Integrating two Complementary Model-based Environments for the Construction of Multimodal Interactive Applications. *Interacting with Computers*, vol. 17, n°3 (to appear), 2006.
- ³² Nedel, L.; Freitas, C.M.D.S.; Jacob, L.; Pi, M.. Testing the Use of Egocentric Interactive Techniques in Immersive Virtual Environments. In proceedings of Ninth IFIP TC13 International Conference on Human-Computer Interaction, 2003, INTERACT'03. Amsterdam: IOS Press, 2003. p. 471-478.
- ³³ Nielsen, J., Mack, R. (eds.) *Usability Inspection Methods* (New York: Wiley) 25-62. 1994
- ³⁴ Nigay L. & Coutaz J. A Generic Platform for Addressing the Multimodal Challenge. *Human Factors In Computing Systems CHI'95 Conference Proceeding*; Denver Colorado USA. 1995: p 98-105.
- ³⁵ OMG. The Common Object Request Broker: Architecture and Specification. In CORBA IIOP 2.2.Framingham 1998
- ³⁶ Ould M., Palanque P., Schyn A., Bastide R., Navarre D., Rubio F. Multimodal and 3D Graphic Man Machine Interfaces to improve Operations. *SpaceOps 2004, AAIA, Spring 2004, Montreal, Canada.*
- ³⁷ Oviatt S. Ten myths of Multimodal Interaction. *Communication of the ACM*. 1999; 42(11):74-81.
- ³⁸ Panttaja, E. M., Reitter, D., Cummins, F. The Evaluation of Adaptable Multimodal Systems Outputs. In Proceedings of the DUMAS Workshop on Robust and Adaptive Information Processing for Mobile Speech Interface. 2004, Geneva Switzerland.
- ³⁹ Paternò, F., Santos, I. Designing and Developing Multi-user, Multi-device Web Interfaces. In proceedings of CADUI 2006 (to appear), Springer Verlag, Bucharest, Romania, June 5-8, 2006.
- ⁴⁰ Reason J. *Human Error*, Cambridge University Press, 1990.
- ⁴¹ Sherry L., Polson P., Feary M. & Palmer E. When Does the MCDU Interface Work Well? Lessons Learned for the Design of New Flightdeck User-Interface. In proceedings of HCI Aero 2002, AAAI Press, pp. 180-186.
- ⁴² Suhm, B., Myers, B., Waibel, A. Model-based and Empirical Evaluation of Multimodal Interactive Error Correction. In Proc. CHI99, Pittsburgh, USA, 15-20 May 1999. pp. 584-591.
- ⁴³ Sy O., Bastide R., Palanque P., Le, D-H and Navarre D. PetShop: a CASE Tool for the Petri Net Based Specification and Prototyping of CORBA Systems. 20th International Conference on Applications and Theory of Petri Nets, ICATPN'99, springer Verlag, p. 145-172.
- ⁴⁴ Vo M.T. & Wood C. Building an Application Framework for Speech and Pen Input Integration in Multimodal Learning Interface. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1996, Vol 6, pp. 3545-3548.
- ⁴⁵ Yee K-P. Two handed interaction on a tablet display. Late Breaking Results, ACM CHI 2004 conference, Vienna Austria, 2004, pp. 1493-1496.
- ⁴⁶ Zhai, S., Barton, A. S., Selker, T. Improving browsing performance: a study of four input devices for scrolling and pointing tasks. Proceedings of INTERACT'97: The IFIP Conference on Human-Computer Interaction, 1997, 286-292.

Type: Oral Presentation

Topic: 5, Design methods and Tools

A Barrier-Approach to Inform Model-Based Design of Safety-Critical Interactive Systems

B.A SCHUPP¹, S. BASNYAT², P.PALANQUE², P.WRIGHT¹

(1) University of York, department of Computer Science, Heslington, York, United Kingdom. Email: {Bastiaan.Schupp, Peter.Wright}@cs.york.ac.uk

(2) LIHS – IRIT, University Paul Sabatier, 118 route de Narbonne, 31062 Toulouse Cedex 4, France. Email: {Basnyat, Palanque}@irit.fr

Keywords: Safety-Critical Interactive Systems, Barriers, Incident and Accident Investigation, Formal Specification Techniques, System Modelling, Human Factors

Abstract

This paper presents a three step approach to improve safety in the field of interactive systems. This approach combines, within a single framework, previous work in the field of barrier analysis and modelling, with model based design of interactive systems.

The approach first uses the Safety Modelling Language to specify safety barriers which could achieve risk reduction if implemented. The detailed mechanism by which these barriers behave is designed in the subsequent stage, using a Petri nets-based formal description technique called Interactive Cooperative Objects. One of the main characteristics of interactive systems is the fact that the user is deeply involved in the operation of such systems. This paper addresses this issue of user behaviour by modelling tasks and activities using the same notation as for the system side (both barriers and interactive system). The use of a formal modelling technique for the description of these three components makes it possible to compare, analyse and integrate them. The approach and the integration are presented on a mining case study. Two safety barriers are modelled as well as the relevant parts of the interactive system behaviour. Operators' tasks are also modelled. The paper then shows how the integration of barriers within the system model can prevent previously identified hazardous sequences of events from occurring, thus increasing the entire system safety.

Résumé

Cet article présente une approche composée de trois étapes pour accroître la fiabilité des systèmes interactifs. Cette approche combine, à l'intérieur d'un seul et même cadre fédérateur, des travaux antérieurs dans le domaine de l'analyse et de la modélisation des barrières, et dans celui de la modélisation des systèmes interactifs.

L'approche exploite tout d'abord SML (Safety Modelling Language) pour la spécification des barrières dont l'implémentation permet la diminution des risques. La façon dont ces barrières sont conçues et spécifiées est décrit dans la deuxième phase et exploite une technique de spécification formelle basée sur les réseaux de Petri appelée ICO (Interactive Cooperative Objects).

Une des caractéristiques fondamentales des systèmes interactifs est liées au fait que l'utilisateur du système est lourdement impliqué dans l'exploitation de ces systèmes. Cet article prend en compte cet aspect utilisateur par la modélisation des tâches et activités des utilisateurs à l'aide de la même que pour la partie système (qui couvre à la fois la partie système et la partie barrière). L'utilisation d'une même technique de description formelle pour ces trois composants offre de nombreux avantages tels que la vérification de la compatibilité des descriptions, la vérification de leur complémentarité ainsi que la vérification du comportement global suite à leur intégration. L'approche est exemplifiée sur une étude de cas du domaine minier décrivant un système d'incinérateur de charbon. La modélisation inclut deux barrières, les parties pertinentes du comportement du système ainsi que les tâches des opérateurs. L'article décrit ensuite comment ces barrières, une fois intégrées dans le modèle du système, peuvent éviter l'occurrence de séquences d'événements pré-identifiées qui pourraient conduire à un incident, accroissant par là même la sûreté du couple opérateur-système.

1. Introduction

Today, safety has become paramount in the design and operation of many technological systems. Often such systems present hazards that cannot be easily eliminated and therefore these systems become safety critical. Safety critical systems can be found in domains, such as in transportation, medicine, industry, and even in financial systems. To mitigate the risk caused by the potential consequences of these hazards, risk reduction must occur for the system to be safe enough to be accepted by society. The means of risk reduction are usually dedicated safety systems that stop the evolution of scenarios leading to unacceptable consequences.

To avoid double use of the word system, we will refer to safety systems as safety barriers, or simply barriers. When we refer to 'the system', this reference is made to the system that is being designed and operated as a whole, for instance the plant, aircraft or computer system. Though a long discussion of safety barriers is beyond the scope of this paper, barriers are usually regarded as systems that prevent or stop an undesired consequence. The ability to stop is important here, and defines the scope of what the barrier is. For instance a fire extinguisher is not a barrier itself, as it has to be operated by a human who must have received some training, and it must be in an easily accessible place. These elements are part of the barrier too. While systems and barriers should be independent to a certain aspect, they will often share components.

In this paper we will deal with a special but very common category of barriers, those that are socio-technical. This means that the barrier is essentially a combination of hardware and software, but also depends on human action for it to function correctly. The barrier thus assigns safety critical tasks to human operators who therefore become crucial in maintaining system safety. As these barriers are sociotechnical, the tasks involve interaction with system software and hardware.

The task of the operator appears often hard to integrate in system design, and may occur too late [1] whilst the technical part of designing a sociotechnical system is often relatively straightforward, correctly specifying and analysing the human tasks and performance appears more difficult. These difficulties may also lead to operators not being aware of a task being safety critical, which can obviously cause accidents.

It is the specification, analysis, verification and documentation of the safety critical human tasks that we are interested in. In this paper we outline an approach that facilitates these tasks. Most importantly, we simplify system analysis by explicitly defining barriers, analysing how these function, and only subsequently integrating them in the system, instead of directly trying to analyse the system as a whole. The length of this paper does not allow a full discussion of our work. We will mention however where we shortened our discussion for brevity.

2. The Approach

The approach employs a formal description technique to provide non-ambiguous, complete and concise models, thus giving an early verification of some potential problems to the designer before the application is actually implemented. However, formal specification of interactive systems often does not address the issues of erroneous user behaviour that may have serious consequences for the system. In order to provide such benefits, formal specification techniques can also be complex, and designers may be reluctant to use them [2]. For these reasons, the ICO approach presented in the paper is tool supported and tutorials, examples and case studies are available through the web site (<http://liihs.irit.fr/petshop>).

We use a three step approach (see Figure 1 for approach overview diagram). Step one uses the Safety Modelling Language (SML) to identify a structure which achieves risk reduction [3]. This structure makes up the safety architecture of the system. Here the specific hazards are analysed, and barriers are devised that can prevent targets (e.g. workers, environment) from being affected by these hazards. In this first step, barriers are treated as black boxes, it is specified why they are in the system, not how they function. Designers are thus supported in reasoning about risk reduction conceptually.

In the second step each individual barrier is analyzed, designed and modelled. Often various techniques are required to achieve this. In this paper we employ the Interactive Cooperative Objects (ICO) formalism [4] based on Petri-nets to model and analyse the mechanisms of the barrier, their specifications and to verify their functions. The result of this step is a full design of the barrier which will achieve the safety function as specified in step one. This may use various parts of the system, hardware, software and human to achieve the required safety function.

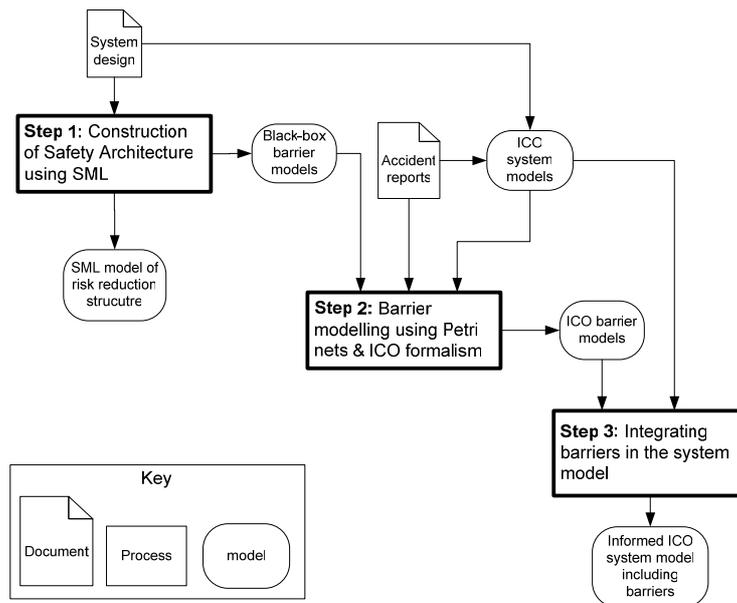


Figure 1. Approach Diagram

In the third step the functions specified by each individual barrier are connected to the system model as a whole by integrating the barrier into the system. In this paper we study this using the ICO-model. This occurs as follows. An operator has a number of functions. Some of these are specified by barriers. Using this mapping, it becomes clear which of the operator's functions are specified by which barrier, and therefore are safety critical. Barrier functions are connected in a similar way to hardware and software components of the system.

2.1 Safety Modelling Language

The first step of our approach uses Safety Modelling Language. In short SML [5] uses the Hazard-Barrier-Target (H-B-T) model to model the safety architecture of a system. The H-B-T model assumes that targets are vulnerable to the effects of hazards, and that targets can be protected against these effects by barriers. In some respects it is similar to other barrier models, such as the accident evolution and barrier function model [6], and the 'Swiss-cheese' model [7]. In Figure 2, a basic example of a SML diagram is shown. It shows that toxic fumes are hazardous to workers. However the worker is protected by a containment system that contains the fumes, thus being a barrier that prevents exposure. As this may not be completely adequate, the worker is further protected by Personal Protective Equipment (PPE). Alternatively prevention is realized by removing the hazard, for example by using a non-toxic substance.

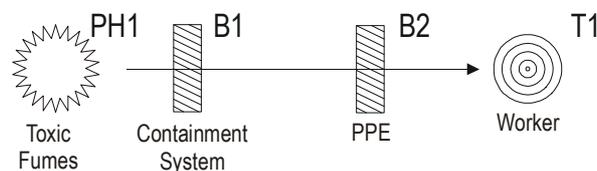


Figure 2. A typical H-B-T diagram. PH1 is a primary hazard symbol, T1 a target symbol, and B1 & B2 are barrier symbols

SML models hazards in a more complex manner than the basic H-B-T model in Figure 2. A hazard is something that has the potential to cause an adverse effect to a target. A hazard is a 'label' that humans apply to complex phenomena perceived as hazardous. SML does not provide insight into the hazardous phenomenon itself but into the relations this phenomenon has with the rest of the design/system. It is modelled using two components: Causal elements that provide a link to the mechanism of the hazard, and effects, that provide the link to the targets. For instance, when the elements 'flammable substance', 'oxygen', and 'ignition source' are present in a design, these will cause a fire hazard, having heat radiation, smoke and high temperature as effects. This is shown in

Figure 3a. An example of a human factors related hazard is a misdiagnosis in interpreting an X-ray photograph in a medical domain. This can for instance be caused by the causal elements ‘training’, ‘available time’, and issues such as ‘X-ray clarity’.

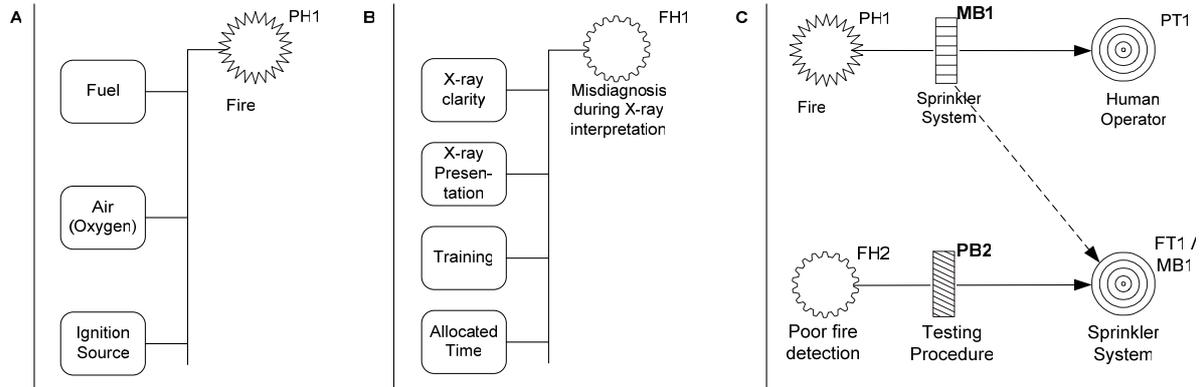


Figure 3: SML representation of (a) fire hazard, (b) a human error hazard, and (c) recursion. FH1 & FH2 are functional hazard symbols, MB1 a mitigative barrier, PB2 a protective barrier

To model the failure of barriers, SML defines *primary* and *functional* hazards. Primary hazards cause direct harm to humans, neighbouring installations, and the environment. The barriers in between primary hazards and primary targets are called primary barriers. Functional hazards are phenomena due to either human factors or other causes that adversely affect other barriers, thus making these fail. Poor fire detection causes a sprinkler system to become inoperable, a testing procedure protects against this, as shown in Figure 3c. In this way, a risk reduction problem is defined recursively; when a barrier is used, it can fail due to a functional hazard

A consequence of this is that the list of primary hazards quickly provides insight into why the systems' safety is critical. Next, accident mechanisms, and the role humans play in these can be understood via recursions. This and many other aspects of the language such as the different kind of symbols and barriers are not further explained in this paper though Figure 3 shows some. For further information on SML see [8].

2.2 System Modelling, Petri Nets, and the ICO formalism

Whilst SML helps to define the barriers and their role in risk reduction, we need to understand which tasks are defined by these barriers, and how they must be integrated in the system. SML is not helpful here. We use the ICO formalism based on Petri nets to achieve that. The ICO barrier models built using the SML model represents both human and system behaviour, thus allowing task analysis. The advantages of the use of formalisms are that they provide non-ambiguous, complete and concise notations. Moreover, they allow to check and prove properties of the design, thus to verify that the barrier will function.

2.2.1 Petri nets

Petri nets are a widely used formal description technique in systems engineering. In this paper, a dialect of Petri nets is used to model both the system and the behaviour of barriers. Brevity prevents a detailed introduction to the Petri net notation however interested readers can look at [9].

Petri Nets are a formalism composed of four elements: the states (called places, depicted as ellipses), state changing operators (called transitions, depicted as rectangles), arcs (relating transitions and places) and tokens (representing the current state of the Petri net).

2.2.2 Informal presentation of the ICO formalism

The Interactive Cooperative Objects (ICOs) formalism is a formal description technique dedicated to the specification of interactive systems [4]. It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance, client/server relationship) to describe the structural or static aspects of systems, and uses high-level Petri nets [10] to describe their dynamic or behavioural aspects.

An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the "input" aspects of the interaction (i.e. how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its "output" aspects (i.e. when and how the application displays information relevant to the user). An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented [11]. The specification can also be validated using analysis and proof tools developed within the Petri nets community. In subsequent sections, we use the symbols in Figure 4.

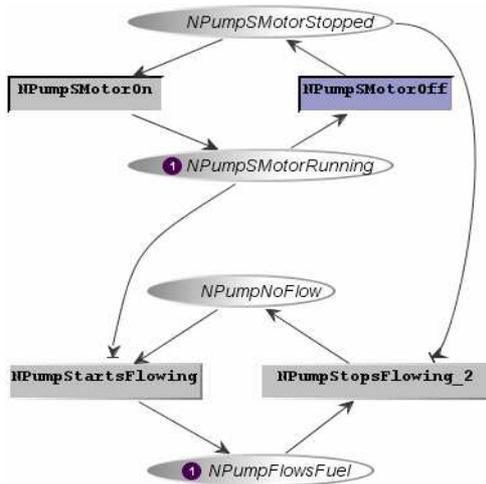


Figure 4. An ICO model of the operation of a pump and its motor

- States of the system are represented by the distribution of tokens into places 
- Actions triggered in an autonomous way by the system are called transitions and are represented as follows 
- Actions triggered by users are represented by half bordered transition 

3. Case Study

We use an example to further explain our approach. It is based on an existing design of a fuel line, which feeds the burners in a cement kiln¹.

We will focus at the start-up procedure of the fuel line. We will analyse why some of these tasks should have been defined as safety critical, and how our method is of help here. We will investigate how using barriers improves design by helping to define safety critical operator tasks. Lastly we will discuss the integration of these barriers with the system.

3.1 System Analysis, starting point

In a real world design situation a natural starting point for our analysis would exist, provided by drawings or other documentation of the design. We therefore shortly discuss the design of the fuel line before proceeding with explaining the proposed method. At this point, hazards and barriers are not yet known, hence we start from a purely a functional description. We will discuss a hazard analysis as well, though this is not strictly part of our method.

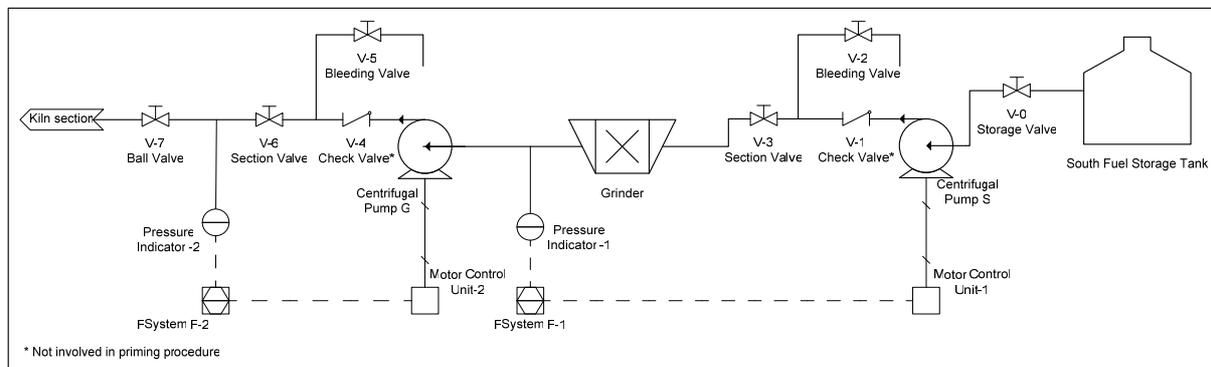


Figure 5: Piping and Instrumentation Diagram of fuel line. See text for explanation.

¹ This case is based on a fatal accident [13] that took place in the US.

The exact manner in which the design is represented is domain dependent. Here we use a Piping and Instrumentation Diagram (P&ID), and a formal model. A formal system model is a required input to our method. Conversely, the P&ID is only required to understand the design, and may be substituted or complemented by other forms of documentation. In a real world situation, the formal model may have to be created before proceeding.

3.1.1 Functional system design

Figure 5 shows the Piping and Instrumentation Diagram (P&ID) describing the Fuel delivery system.

V-1 and V-4 represent check valves, V-2 and V-6 are section valves, V-3 and V-5 are for bleeding, and V-7 is a ball valve used to control the start-up procedure. Bleeding is done before system start-up to make the pipes free of air, and to prime the pumps. Without priming, the pumps cannot create suction, and thus do not pump. MCU-1&2 are the Motor Control Units, PI-1&2 are Pressure Indicators, FSystem-1&2 are the implementations of the pump control loops.

3.1.2 Formal system description

The formal system description is achieved using the ICO formalism described in section 2.2. The model is here only briefly discussed, see [12] for a more in depth discussion.

In [12], we have modelled each individual component of the plant (e.g. pumps, grinders, fuel tanks etc) using the ICO formalism. These individual components have been interconnected based on fuel flow, because that most accurately describes how the process functions. With this configuration, we are able to see what happens to fuel if for example a motor is not turned on, or a valve is left open etc. Figure 6 illustrates the complete system model for the fuel system. The complete model can be found on our project ADVISES web site <http://www.cs.york.ac.uk/hci/ADVISES/paperSBPW>. For explanatory purposes, the components have been grouped and labelled.

- | | |
|--|---|
| A) Fuel tank | F) Pump-G Motor and corresponding fuel flow |
| B) Pump-S Motor and corresponding fuel flow | G) Pump-G Bleeding Valve (V5 in Figure 5) |
| C) Pump-S Bleeding Valve (V0 in Figure 5) | H) Pump-G Section Valve (V6 in Figure 5) |
| D) Pump-S Section Valve (V3 in Figure 5) | I) Ball valve (V7 in Figure 5) |
| E) Grinder Motor and corresponding fuel flow | J) Plant Kilns |

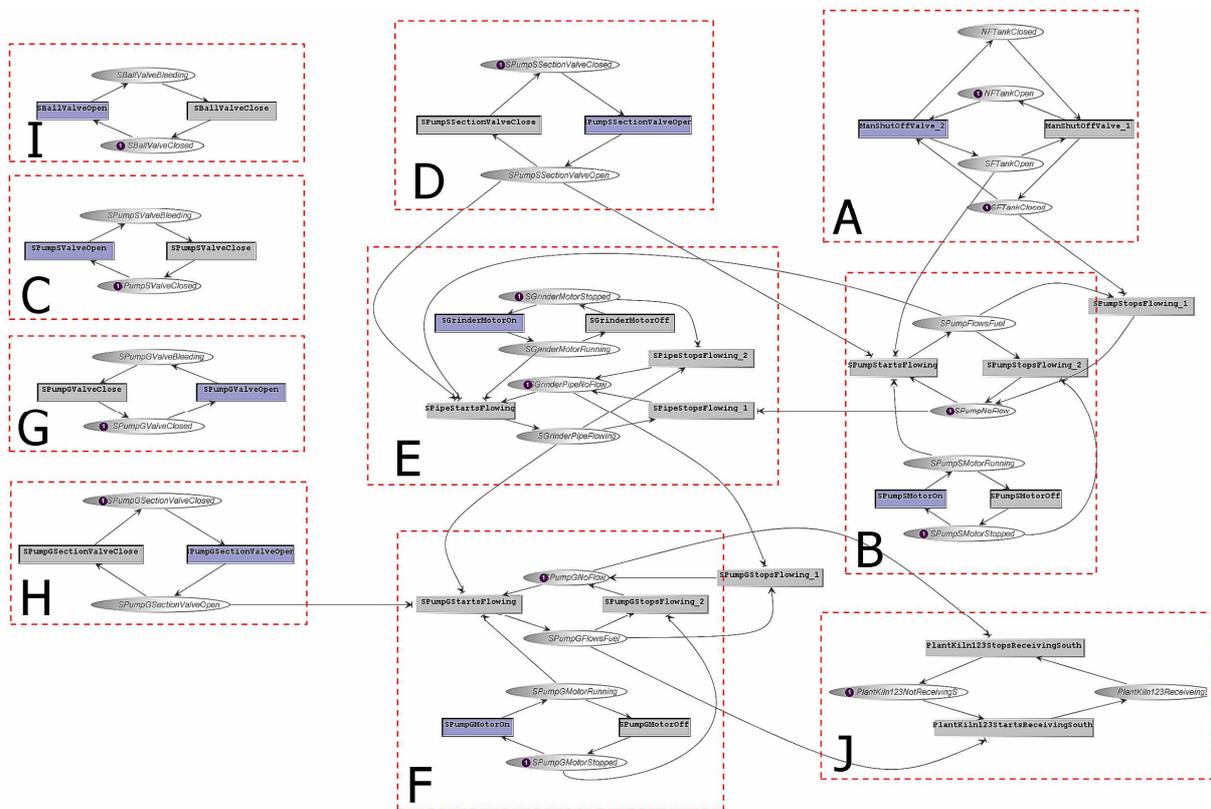


Figure 6. Petri net system model of the fuel line

3.2 Step 1: HBT analysis

In this example hazard identification is relatively straightforward. A group of experts using a common identification method (e.g. a Hazop) would for instance quickly realise that both too much static pressure and pressure waves which may compromise containment can occur in this system. Though other hazards might be identified as well, we will concentrate on these pressure related hazards here. Our method however becomes important after hazard identification. Now two steps must be taken. Firstly, potential consequences and risk must be estimated, then barriers to reduce the risk must be designed. Though both steps are supported by the method, this paper focuses at designing the barriers, and in particular at integrating and understanding tasks carried out by humans as part of these barriers. In this example, typical barriers may include surge arrestors, emergency shutdown or procedural barriers (e.g. limiting pump power during start-up). The method helps to select, specify and verify these.

In Figure 7 the results of a Hazard-Barrier-Target analysis using SML are shown. The primary hazard is a fire hazard, as this will cause harm to for instance, operators. Many potential barriers are available that may stop them from receiving such harm. Fire normally is caused by the presence of fuel, air and an ignition source. In this case fire is prevented by an Inherent Barrier (IB1), containment, which keeps the fuel separated from air and ignition sources. In case containment (IB1) fails, and fire occurs, a sprinkler system is present to mitigate the effects of the fire. Figure 7A shows a SML representation of this. More complicated and precise models are probably appropriate here, for instance to include failure modes of the barrier (e.g. spraying fuel or just leaking). SML facilitates these, but for brevity we will omit further discussion.

Containment thus is an important barrier here. Our further discussion focuses on protecting this barrier against functional hazards. That is, the integrity of pipe work and casing of equipment such as the pumps and the grinder must remain. This may however be compromised by high pressure in the system. Therefore pressure surges must not occur in the system. In other words, these are functional hazards. A pressure surge may for instance occur because of starting a pump in the wrong way. For this to happen, four causal elements must be present; the pump must be running, but it must contain air as is shown in Figure 7B. Then the air must be bled from it, which will cause sudden flow. In an inelastic system, this will cause a pressure surge. Hence, bleeding the pump (also called priming; fill it with liquid as it cannot otherwise create suction), and starting the pump must not occur in the wrong order. Two alternative causes of high pressure are possible as well; water hammer due to sudden stop of flow, and static high pressure, see Figure 7C.

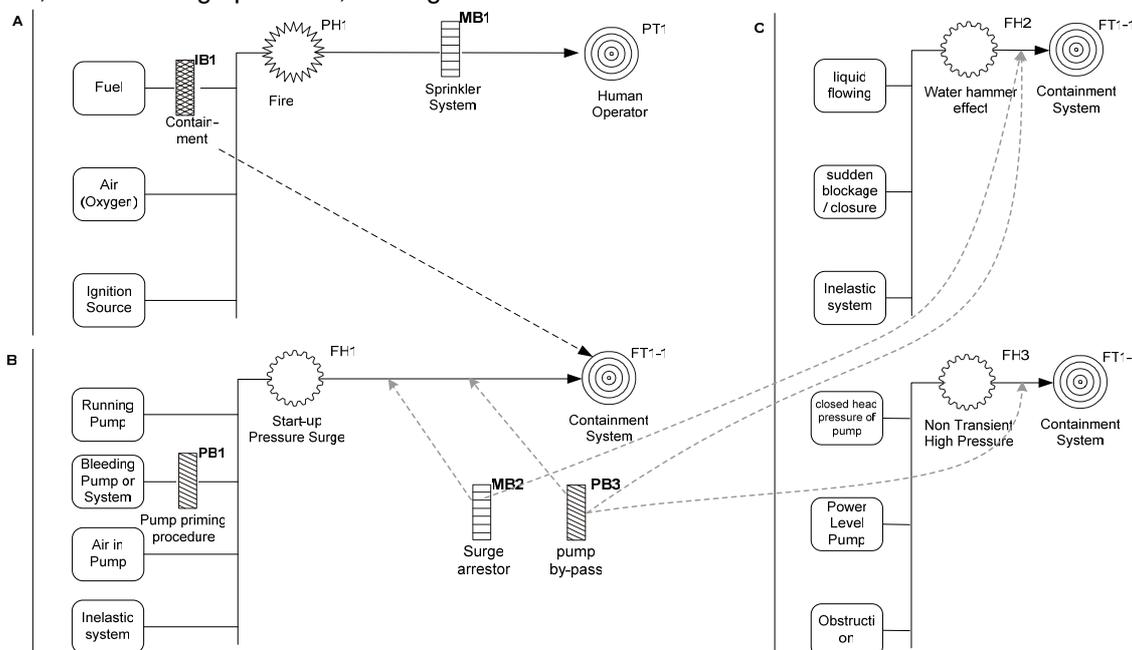


Figure 7. SML representation of hazards, barriers and targets discussed in this text.

The system designers may now evaluate alternative barriers that potentially reduce the risk of a pressure surge. SML notation allows system designers to conceptually understand which barrier is best to use. Although PB1, the pump priming procedure, will prevent pressure surges (FH1) altogether, MB2 will protect against most transient pressure waves (FH2), now also including water

the pump motors must remain off until the procedure defined by PB1 is completed. This is simple to implement in the barrier model, however in practice this can be more complicated, and detailed discussion of this is beyond the scope of our discussion. It may perhaps be achieved using signs near the switches referring to the procedure.

We have modelled PB1 using the ICO formalism - see Figure 8. The diagram has been segregated into several sections for explanatory purposes. The model of this barrier is a combination of hardware and human actions. The hardware concerns the three motors, modelled in Figure 8-1, the Pump-G motor, Grinder motor and Pump-S motor. It also concerns the valves. However these are modelled together with their interaction with the required operator's actions in the remainder of the figure.

The barrier is mainly made up of arcs connecting transitions and places rather than the transitions and places themselves. It can be decomposed into three main parts. The first part concerns preventing the three motors (part 1 of Figure 8) from running before the procedure is completed. This is modelled by means of test arcs which impose a pre-condition on operating the bleeding valves (V2 and V5) or the section valves (V3 and V6) preventing them from being opened during the priming procedure if the motors are running.

The second part of PB1 is the obligation of order of events for the operator's interaction with the system. For instance part 5 shows the opening of the 1st section valve which cannot be opened before air has been bled from the first section of piping (part 3).

The third part is also an obligation of order. It prevents the operator from closing the section valves (as shown in part 9) before fuel arrives at the plant kilns (a token is set into place *PlantKiln123ReceivingFuel*). Once fuel arrives at the plant kilns, the task is complete. The closing of the section valves (step 9 in the procedure) is optional and depends on whether the fuel delivery system will be started immediately in which case the section valves are left open, or later, in which case they can both be closed.

Now the barrier model is established it can be analysed further. For instance human factors methods can be used to understand whether humans can achieve the tasks that the barrier specifies.

3.4 Step 3: Connecting technical and human barriers in the system model

Integrating the barrier in the system in the real world obviously involves much more than just integrating the ICO models. As discussed, the initial state required by PB1 might be implemented by adding a sign to a power switch on pump, which refers to the procedure specified by PB1. Such implementation issues are beyond the scope of this paper however, and require additional methods. Here we only have space to briefly discuss what happens to the ICO models.

The initial ICO system model as presented in Figure 6 significantly changes because of the connection of PB1. For space and readability reasons we cannot include the extended system model in this paper the interested reader can find it on the website mentioned earlier. The complexity of the model has significantly increased, mainly because of the number of added arcs. Also several places and transitions have been added for instance to represent the existence of air in the pipes which was previously not represented in the system model as this has nothing to do with the initial functional specification of the system.

Two further issues must be noted. Firstly the addition of the barrier to the system model must not change the behaviour of the system, except of course for excluding the hazardous state it is designed for. That is to say, an action or task that was previously available on the system side or via interaction with the user must not be changed and must be still available. Secondly, the actual modelling of the barrier inevitably involves direct integration with the system model because the barrier not only takes existing components, but may also rely on adding extra arcs to existing components. Hence it is not possible to model the barrier without taking notion of the system model; barriers exploit a subset of the system model, sometimes with extra transitions and places. Thus step 3 of the approach is the addition of the complete barrier, including any additional components that were not present in the existing system model.

Using the ICO model representing the system behaviour, it becomes possible to verify that the system still works, to prove that the hazardous state is no longer reachable and to analyse in which way the newly integrated barrier may fail.

4. Conclusion and future work

This paper presented a three-step approach for identifying, modelling and formally specifying safety critical human tasks, interactive system and their associated barriers. The Safety Modelling Language is used to model existing as well as identify new socio-technical barriers. We then used the Interactive Cooperative Objects (ICO) formalism to specify the behaviour of the barriers. Finally we integrate these barriers with a system model, also specified using the ICO formalism.

The approach presented is part of a larger framework of research centred on model based design, aiming to improve the design of safety critical interactive systems by accounting for errors (technical and human related) early in the design process. We believe that by identifying and incorporating socio-technical barriers such as those discussed in this paper within their relevant models, we can obtain an early verification of some potential problems before the application is actually implemented. This will ultimately lead the design of safer safety critical interactive systems by embedding reliability, efficiency and error-tolerance in the system. For instance, as part of this larger framework, we have shown in previous papers [3] how system design can be improved by using accident investigation techniques.

Acknowledgements

This work was supported by the EU funded ADVISES Research Training Network, GR/N 006R02527. <http://www.cs.york.ac.uk/hci/ADVISES/>

References

- [1] Daouk, M, Leveson, N. G., An Approach to Human-Centred Design, Workshop on Human error and System Dev., Linkoping, Sweden, (June 2001) <http://sunnyday.mit.edu/papers.html>
- [2] Bowen J. P., Hinchey M. G., High Integrity System Specification and Design, Springer-Verlag, London (1999)
- [3] Schupp, B. A., Smith, S., Wright, P. and Goossens, L. Integrating human factors in the design of safety critical systems - A barrier based approach, Human Error, Safety and Systems Development (HESSD 2004), W. Johnson and P. Palanque (Eds.), Vol. 152 (2004) 285-300, Springer.
- [4] Bastide, R., Sy, O., Palanque, P., Navarre, D., Formal specification of CORBA services: experience and lessons learned. ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'2000). ACM Press; 2000.
- [5] Schupp, B. A., Lemkowitz, S. M. L., et al. (2001). Application of the Hazard-Barrier-Target (HBT) Model for More Effective Design For Safety in a Computer-Based Technology Management Environment. CCPS ICW: Making Process Safety Pay: the business case, Toronto, AIChE/CCPS.
- [6] Svenson, O., The Accident Evolution and Barrier Function (Aeb) Model Applied to Incident Analysis in the Processing Industries, Risk Analysis 11(3): 499-507 (1991).
- [7] Reason, J. T. Human Error. Cambridge, Cambridge University Press (1990).
- [8] Schupp, B. A., Hale, A. R., Pasman, H. J., Lemkowitz, S. M. L., Goossens, L., Design support for the systematic integration of risk reduction into early chemical process design, Safety Science (2005, in press) 2&
- [9] Petri, C. A. Kommunikation mit automaten, Technical University Darmstadt (1962)
- [10] Genrich, H.J. Predicate/Transitions Nets, High-Levels Petri-Nets: Theory and Application. K Jensen and G Rozenberg (Eds) Berlin: Springer Verlag (1991) pp 3-43
- [11] Navarre, D., Palanque, P., Bastide, R., and Sy, O. Structuring Interactive Systems Specifications for Executability and Prototypability, 7th Ergonomics Workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'2000, Limerick, Ireland, Lecture Notes in Computer Science, no. 1946. Springer (2000), 97-109
- [12] Basnyat, S., Chozos, N., Palanque, P. Multidisciplinary perspective on accident investigation. special edition of Elsevier's Reliability Engineering and System Safety journal. Complexity in Design and Engineering Workshop Journal Special Edition, Elsevier(2005)
- [13] United States Department Of Labor Mine Safety And Health Administration Report Of Investigation Surface Area Of Underground Coal Mine Fatal Exploding Pressure Vessel Accident January 28, 2002 At Island Creek Coal Company Vp 8 (I.D. 44-03795) Mavisdale, Buchanan County, Originating Office Mine Safety And Health Administration District 5 P.O. Box 560, Wise County Plaza, Norton, Virginia 24273 (2002)

Investigation of Structural Properties of Hazard Mitigation Arguments

Mark A. Sujan

Department of Computer Science, University of York, UK

Michael D. Harrison

Informatics Research Institute, University of Newcastle upon Tyne, UK

ABSTRACT: Arguing that a system is sufficiently safe to operate in a particular context is an important element in the development of safety-critical systems. Hazard mitigation arguments support safety claims by providing evidence. They do this, typically, by appealing to barriers intended to prevent and to protect from a specific hazard. The paper summarises work on the structure of arguments, and then addresses two research questions: how understanding of the quality of arguments can be applied to hazard mitigation arguments; how making the underlying implied barrier model explicit can improve the argumentation and provide useful insights for managing risks. The EUROCONTROL Reduced Vertical Separation Minima Functional Hazard Analysis (RVSM FHA) is used to demonstrate these ideas.

1 INTRODUCTION

Argumentation is an important part of the development of safety critical systems. It provides information about why a system can be assumed to be sufficiently safe, and it may convey a measure of confidence. In many safety-critical industries such information is documented in a safety case.

Increasingly, incremental approaches to safety case development are employed and are mandated by standards such as UK Def-Stan 00-56 (UK MoD, 2004) or the Eurocontrol Safety Regulatory Requirement 4 (Eurocontrol, 2001c). The analysis that lies behind such argumentation may assist system design during the early stages of the life-cycle, for example by outlining the principal structure of the safety argument and the type of evidence required (Kelly, 2001). This can avoid the need to re-design parts of the system late in the life-cycle on realising that no adequate safety argument can be constructed for the proposed design.

Toulmin's work on arguments in general, and developments relating to safety arguments in particular, have emphasised the importance of structure in argumentation. It is now generally accepted that safety arguments should have a structure consisting of claim, argument, and evidence. A graphical notation has been developed (GSN) (Kelly, 1999), to facilitate construction and communication of structured safety arguments. Explicit reasoning about the structure and the quality of arguments leads to notions such as confidence, rigour, coverage, uncertainty, depth and breadth. Structural analysis of

safety arguments based on these concepts may provide useful insights for the development and assessment of safety arguments.

Hazard mitigation arguments operate at a lower level of system detail. In a safety case they provide evidence to support claims that all hazards have been mitigated, that training requirements have been identified, and so on. They have a risk-based structure, and they appeal to barriers with understood and verifiable mitigation characteristics as their evidence. They are derived through a multidisciplinary social negotiation process, and as a result do not assume (deliberately) a single underlying background or model.

In this paper we consider: (a) how the concepts and notions derived from general considerations of argument quality can be interpreted and applied usefully to the specific category of hazard mitigation arguments; (b) the utility of making explicit the implied (barrier) model of system safety.

The structure of the paper is as follows. In section 2 we consider issues of structure and quality in safety arguments. We then discuss (section 3) the role that barriers can play in providing mitigation against potential hazards. These ideas (section 4) are then considered in the specific context of the RSVM hazard analysis. A discussion (section 5) concludes the paper.

2 STRUCTURAL CHARACTERISTICS OF ARGUMENTS

2.1 The role of structure in arguments

Consider the simple example depicted in Figure 1 arguing that the reduction of vertical separation between aircraft has a positive effect on the performance of air traffic controllers¹.

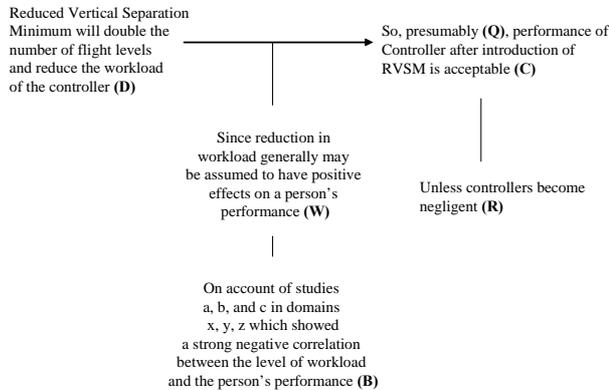


Figure 1: Elements of an argument

Toulmin (Toulmin, 1958), in analysing the structure of arguments in general, identified six components that contribute to an argument's well-formedness thereby facilitating its assessment. The *claim C* is the property or statement that is asserted and argued for ("Performance of Controller after the introduction of RVSM is acceptable"). To support this claim specific *evidence D* ("RVSM will double the number of flight levels and will thus reduce workload of the controller") is produced that should relate to the claim. The *argument* or *warrant W* explains how the evidence supports the claim. The relationship between claim and evidence is made explicit as rules, principles, inferences and so on ("Reduction in workload generally may be assumed to have positive effects on a person's workload"). The warrant can itself be backed by specific evidence referred to as *backing B* ("Studies, which have shown a strong negative correlation between the level of workload and the person's performance"). The backing used to support the warrant consists of concrete, factual information. The warrant, on the other hand, posits a general and practical rule explaining how, given these facts, certain evidence may be used within an argument to support specific claims. Since it is the case that the warrant need not necessarily authorize the step from evidence to claim unconditionally, a *qualification Q* on the strength of its inference needs to be provided ("Presumably"). Finally, known exceptions or *rebuttals R* should be explicitly stated ("Unless controllers become negligent").

Dependability arguments are no different structurally than other types of arguments. The claims in safety arguments often represent safety requirements, safety objectives, target levels of safety or derived sub-claims. Dependability arguments include a wide range of different pieces of evidence including results from HAZOP studies, Fault Trees, qualitative descriptions of the processes that have been followed, descriptions of procedures and training materials. However, in order to engender confidence in the argument it is important to express the relationship between all these pieces of evidence and the claims they are intended to support *explicitly*.

It is also critically important that the context, within which dependability arguments are developed and expressed, is presented explicitly as well as a statement of assumptions that have been made. Contextual elements include definitions of systems or components under consideration, definitions of terms such as "tolerable" and "negligible", descriptions of standards followed and so on.

Assumptions also deserve special attention. They are often implicit rather than explicit in the argument. These assumptions may concern the safety of a system prior to a planned modification, or may concern the independence of components or events for example. Assumptions also make explicit rebuttals, such as from Figure 1, which could be expressed as "Assumption 1: It is assumed that controllers monitor traffic as vigilant as before the introduction of RVSM."

Goal Structuring Notation (GSN) (Kelly, 1999) explicitly represents the elements of importance in dependability arguments (claims, evidence, assumptions, justifications, context etc.). It also makes explicit the relationship between these elements by describing graphically how claims are broken down into sub-claims until the point where evidence is provided to support the claims. The strategy and rationale adopted in the argument development are also represented via strategy and justification elements. GSN retains all the essential elements identified by Toulmin, while at the same time facilitating the formulation, presentation and assessment of dependability arguments.

2.2 The quality of an argument

The aim of safety arguments (or arguments in general) is to instil confidence in a third party that a set of particular claims is true². Confidence in an argument can be increased by ensuring that the evidence (Govier, 1988):

- is acceptable or true

² There are, of course, further aims, such as supporting the development process by pointing out safety-related issues early on etc.

¹ Note that this argument is a hypothetical example created for the purpose of illustration.

- is relevant to the claim
- taken together, provides sufficient grounds to move to the conclusion.

Uncertainty on the other hand can arise from:

- uncertainty attached to the evidence (for example, experimental assessments of workload levels),
- uncertainty attached to the warrant or argument (for example, the basic rule that a reduction in workload results in improved performance),
- the coverage of the evidence (for example, a reduction in workload by itself might not be sufficient to claim that controller performance is acceptable).

Dependence of the pieces of supporting evidence on one another is also an important aspect of the structure of an argument that can be analysed. Govier, when describing “Support Pattern Types” (Govier, 1988), makes a distinction between single, linked and diverse (‘convergent’ in Govier’s terminology) argument support. A *single support* type implies that a claim is supported by a single argument (i.e., a single evidence-warrant-backing structure). A claim may also be supported interdependently by a number of arguments, where each argument’s support rests on the validity of the other arguments (*linked support*). Finally, a number of arguments may also support a claim independently of one another (*diverse support*). This qualitative description of argument structures has limitations as it does not answer questions, such as *how much* confidence can be justifiably placed in an argument. Bloomfield and Littlewood have argued that a formal theory of uncertainty based on probability is required to assess these issues, and they have provided a tentative formalism (Bloomfield & Littlewood, 2006). As this paper is concerned with general structural concepts, no quantitative probabilistic framework is employed.

The general structure of arguments may be used to derive generic ways of strengthening specific arguments or to increase confidence in their validity (see Sujan et al., 2006 for a more extensive discussion). Increasing the depth of arguments can address uncertainty related to the *rigour* demanded by the third party. For example, in order to increase confidence in the evidence that RVSM leads to a reduction in workload, a further argument could be constructed, taking the original piece of evidence as a claim in its own right. Additional evidence, such as reference to an experimental assessment of workload under conditions comparable to those expected under RVSM could be used to support this claim (which was previously treated as evidence). Depth approaches ‘explain better’ (or in more detail) the argument, thereby increasing confidence, and poten-

tially also pointing out hidden assumptions and other problems. In the example above, such an assumption would be the expectation that the experimental assessment of workload is indicative of workload levels experienced during actual operation.

To address *uncertainty inherent in the evidence or in the warrant*, the breadth of an argument should be increased. For example, even though experimental workload assessments may be indicative of workload experienced in real environments, it is not clear that this will be true in the proposed context. There is inherent uncertainty attached to this kind of evidence. Breadth approaches give diversity to the argument and to the evidence. Diverse evidence could consist of the reference to statistics from the experiences of RVSM in the transatlantic airspace, where this mode of separation management has been operational for many years. The characteristics of the transatlantic airspace are different from the characteristics of the European airspace, and may therefore lead to conjecture as to whether these statistics can be transferred. However, in conjunction with the experimental workload assessment, the auditor may now entertain a higher degree of confidence in the overall claim.

A common approach to arguing for the dependability of a system in the context of a breadth-approach is by means of a ‘product-leg’ and a ‘process-leg’. However, as pointed out in (Bloomfield & Littlewood, 2006) it may often be the case that different argument legs are not independent or fully diverse, and this poses a problem in determining the confidence that can be placed in the argument.

2.3 Hazard mitigation arguments

Safety-critical systems in the contexts assumed here require systematic risk assessment with an associated mitigation of all relevant risks to acceptable levels. The evidence provided by risk assessment is an essential building block of a safety argument or safety case. It may be presented as evidence to the claim that all hazards have been sufficiently mitigated, as well as evidence to other claims, such as the claim that all relevant training requirements have been identified etc.

Hazard identification and assessment is frequently performed with the aid of structured techniques such as HAZOP (Kletz, 1992) and FMEA (IEC, 1985), or more recently, comparable techniques taking into account specifically human activities, such as PHEA (Embrey, 1992), or TRACER (Shorrock & Kirwan, 2003). These techniques are typically applied by groups in sessions that bring together participants with different backgrounds and areas of expertise. In an aviation setting, participants could include air traffic controllers, pilots, engineers, risk managers, and human factors experts.

The output of such group sessions is often recorded in tabular form, for example HAZOP tables, specifying the applied guide word, the description of the hazard, possible causes, expected consequences, as well as possible mitigation mechanisms. As the systems under consideration are becoming increasingly complex (or as their complexity is increasingly being recognised), reasoning about hazard mitigation often turns into a more complex argument in its own right, where the simple tabular form may not be an adequate representation anymore (in particular when socio-technical aspects are considered).

Hazard mitigation arguments have a risk based structure. Their overall strategy is to demonstrate that the risk associated with a particular hazard is tolerable by breaking this claim down into the two components of risk, namely the probability of occurrence of the hazard and severity of the consequences. Both these argument legs appeal to barriers to demonstrate that the respective claims hold.

The next section takes a closer look at barriers, and investigates how the notions of depth and breadth should be interpreted in the context of hazard mitigation arguments, as well as how the barriers appealed to could be made explicit.

3 BARRIERS

3.1 Barrier concept

Hollnagel (Hollnagel, 1999) defines a barrier as an obstacle, an obstruction or a hindrance that may either (a) prevent an action from being carried out or an event from taking place, or (b) thwart or lessen the impact of the consequences. Hazard mitigation arguments appeal to barriers to demonstrate that the probability of occurrence of a particular hazard has been reduced (preventive barrier) or that the severity of the consequences of the hazard has been contained (protective barrier). Hollnagel further distinguishes between the function that a barrier fulfils and the system providing this function (barrier system). Barrier functions could involve the prevention of a particular hazard or the protection from the hazard's consequences. Barrier systems, on the other hand, can be classified in the following way (Hollnagel, 1999):

- *Material barrier:* A barrier that prevents a hazard or protects from a hazard through its physical characteristics, e.g., a physical containment protecting against the release of toxic liquid.
- *Functional barrier:* A barrier that prevents a hazard or protects from a hazard by setting up certain pre-conditions which have to be met before a specific action can be carried out or before a specific event can take place, e.g., a door lock requiring a key, or a logical lock requiring a password.

- *Symbolic barrier:* A symbolic barrier requires an interpretation by an agent to achieve its purpose. Examples include all kinds of signs and signals.
- *Immaterial barrier:* A barrier that has no physical manifestation, but rather depends on the knowledge of people. Examples include rules or expected types of behaviour with respect to a safety culture.

As safety-critical systems are increasingly being understood in terms of the fact that they are large, complex socio-technical organisations, the barriers that are being described become socio-technical systems themselves. Many barriers now take the form of a person (or several people) interacting with equipment or advisory systems and relying on procedures. For example, the *Lost Communication Procedure*, used when an aircraft is not fulfilling the required equipment standard in the RVSM space because of a communication equipment failure, defines actions to be carried out by the air traffic controller, as well as by the aircraft crew with their respective supporting technology. Hence the barrier, abbreviated as Lost Communication Procedure, comprises many socio-technical aspects (and further barriers at lower levels of abstraction).

3.2 Depth and breadth

As descriptions of hazard mitigation can be regarded as arguments, all the concepts identified above that relate to the quality of arguments should apply. Adding depth to an argument increases confidence by explaining in more detail a particular piece of evidence, and by pointing out assumptions and dependencies. In the case of hazard mitigation arguments where an appeal to barriers is involved, it may be useful to equate the notion of depth with the level of abstraction of a barrier. This implies (considering the example above) that confidence in the mitigation argument for communication equipment failures could be increased by specifying in greater detail at lower levels of abstraction the Lost Communication Procedure. This would not mitigate any uncertainty inherent in the procedure (such as the probability that the procedure will not be followed etc), but it would none-the-less increase the confidence in the overall argument. Measuring the depth of an argument does not provide any measure of the 'quality' of such an argument. However, it does provide an indication as to the level of abstraction and rigour used throughout the argument.

Adding breadth to hazard mitigation arguments implies appealing to diverse barriers. In this way, any uncertainties relating to the success of a particular barrier can be mitigated. For example, if there is uncertainty as to whether the Lost Communication Procedure sufficiently mitigates equipment failures, additional barriers, such as the introduction of compulsory reporting points at the entry and exit of

RVSM space could be proposed. Such reporting points would allow air traffic controllers to reason about the position of an aircraft in case of communication equipment failures. However, as in all cases where diversity is used, it is very difficult and important at the same time to assess any kind of dependencies that may exist between the diverse elements. In this particular example, it may be assumed that the success of the calculation of the air traffic controller will depend on the pilots' following the Lost Communication Procedure (e.g., not to deviate from previously assigned flight paths etc).

3.3 *Explicit model*

The evidence provided in a safety argument or a safety case is derived from a large number of sources and from various underlying models. In particular, the evidence provided by hazard mitigation arguments usually stems from multi-disciplinary group discussions and negotiation processes, where different perceptions and models are invoked. This is done on purpose, in order to make best use of the different experiences and the different expertise of all the relevant stakeholders. As such, there is no unifying underlying model of whole system safety. However, the safety argument implicitly defines a structure for describing how these barriers are used in mitigation. This structure describes relationships between barriers both temporal and logical.

Temporal order describes whether a barrier is intended to prevent a hazard or to protect from its consequences (and it describes temporal order within these categories). Order also describes different degrees of mutual dependence, in particular simple logical relationships. Barriers may prevent a hazard or protect from its consequences interdependently by forming a logical AND-relationship. They may also perform the function of prevention or protection independently (thus forming an OR-relationship). It is also possible that a barrier is the only preventive or protective obstacle for a particular hazard. These idealised relationships ignore the different degrees of dependence and relevance of each barrier, but can none the less serve as the basis for further analysis. Making the implied barrier model explicit may have a number of benefits for safety case developers and risk managers. The activity of defining the barrier model explicitly requires the developer to define the relationship between the barriers within an argument, as well as the relationship to barriers appealed to in other arguments. Further anticipated benefits include:

- Possibility to check for inconsistencies and contradictions
- Identification of potentially weak configurations
- Verification of the configuration in actual practice

- Assistance with planning future changes and with assessing the impact of proposed changes

The first issue refers to the fact that hazard mitigation arguments are not usually considered together, but rather during the process of hazard identification and assessment, one after the other (even though there may be some thought during the mitigation stage as to how a barrier may mitigate several hazards for efficiency and economic reasons). However, in order to pick up inconsistencies or even possible contradictions (within the assumptions) of different mitigation argument, these would need to be explicitly related to one another (e.g. how does the introduction of compulsory reporting points interact with other newly introduced procedures or assumptions about workload made elsewhere?).

The second issue is concerned with the identification of potentially weak configurations, such as single barriers guarding against high-risk hazards or multiple hazards.

As was discussed, barriers may be described at varying levels of abstraction, and in practice they are realised through a combination of human, technical and organisational resources. An explicit representation of the assumed or proposed barriers can serve as the basis for an evaluation of how these perform in actual practice (e.g. how is the barrier of compulsory reporting points actually put in place, what kind of interactions does it entail, what kind of equipment is used, what kind of representations are invoked?).

Finally, an explicit representation of barriers that have been appealed to may also serve as a useful tool in planning changes and in assessing the impact of proposed changes. The tight integration of processes and activities in complex socio-technical systems means that the impact of changes on activities that are immediately affected cannot be assessed properly within their local context. Assumptions and hidden dependencies may influence and conflict with assumptions made elsewhere. A model making explicit all the barriers appealed to and the various assumptions made, could facilitate the assessment of both direct and secondary effects of change.

4 RVSM FUNCTIONAL HAZARD ANALYSIS CASE STUDY

4.1 *Description of RVSM FHA*

RVSM is an EATMP programme established to contribute to the overall objective of enhancing capacity and efficiency while maintaining or improving safety within the European Civil Aviation Conference (ECAC) airspace. The main scope of RVSM is to enhance airspace capacity. The introduction of RVSM will permit the application of a 1000 ft vertical separation minimum (VSM) between suitably

equipped aircraft in the level band FL290 - FL410 inclusive. Before the introduction of RVSM the VSM was 2000 ft (referred to as CVSM).

A prerequisite for the introduction of RVSM was the production of a safety case to ensure that minimum safety levels were maintained or improved. The Functional Hazard Analysis (FHA) constitutes an essential part of the Pre-Implementation Safety Case (PISC). The FHA document which forms the basis for the study of this section is publicly available (Eurocontrol, 2001a) as is the Pre-Implementation Safety Case (Eurocontrol, 2001b).

The availability of these documents is a welcome opportunity for the research community to demonstrate and to assess their theoretical concepts and tools. It should be pointed out that the analysis performed in this paper is not intended as a criticism of the way the RVSM safety case and FHA were developed. Rather, the FHA simply serves as an example to demonstrate and to contextualise some theoretical principles.

Three areas have been considered in the FHA:

1. Mature / Core European air traffic region (EUR) RVSM area
2. Mature / Transition space
3. Switchover

For each area a number of scenarios were created for the FHA sessions. In total 72 valid hazards have been analysed during the FHA and safety objectives have been established for each of them. The report concludes that 70 hazards achieved their safety objectives but that two hazards were assessed as safety critical and not tolerable.

In the analysis below the FHA Session 1 / Scenarios 1 and 2 are considered. Session 1 was concerned with the identification and analysis of hazards relating to the core EUR RVSM airspace focussing on both ground-related and airborne hazards. These two sessions identified 30 valid hazards. In the analysis below 15 hazard mitigation arguments are analysed.

The FHA arguments in the document are provided in textual form which makes it difficult to analyse and describe structure and dependencies precisely. The top-level arguments of the Pre-Implementation Safety Case (PISC), of which this FHA is a part, have been articulated fully in GSN.

All hazard mitigation arguments follow the same top-level structure: the claim that the risk arising from a hazard is tolerable is broken down into a claim that the severity is at most x , and a second linked claim that the probability of occurrence of this hazard is not greater than y .

4.2 Structural analysis: depth and breadth of arguments

Structural analysis proceeds by investigating the depth and the breadth of the arguments conducted separately for both the severity and the probability branch. Different support pattern types were identified to consider breadth. The depth and breadth of arguments were measured (somewhat arbitrarily) by considering the deepest and broadest paths respectively.

Table 1 shows that out of the total of 30 arguments (i.e., 15 arguments each consisting of a severity and a probability branch), the majority (17) possesses a depth of 2 and the maximum depth of any argument is 4.

Table 1: Analysis results showing distribution of depth, breadth, and support type (Single, Linked, Diverse) in the probability and severity branches

	Depth (#30)				Breadth (#30)					Support (#74)		
	1	2	3	4	1	2	3	4	7	S	L	D
Probability	5	8	2	-	5	8	-	1	1	20	-	17
Severity	-	9	5	1	8	5	2	-	-	29	5	3
Σ	5	17	7	1	13	13	2	1	1	49	5	20

On the severity side, the most common form of argument (possessing a depth of 2) is the claim that the severity is at most Cat. x , supported by a description of the safety implications giving rise to a Cat. x classification. This, in turn, is supported by a description of operational consequences (see Figure 2). This would be recorded as a Depth 2 – Breadth 1 – Single Support – Single Support Argument (D2-B1-S-S). Greater breadth is used for a linked description of operational consequences, and for appeal to barriers. In the cases analysed there is one instance of linked barriers and one instance of diverse barriers.

Probability arguments typically take one of two forms: either the claim that the probability of occurrence of a hazard is at most x is directly supported by expert judgement (D1-B1-S), or this claim is supported by first noting that the probability currently is $y > x$ (i.e. larger than required), but that it will be reduced to x . The former claim is supported by expert judgement as before, while the latter claim is supported by appeal to one or more barriers (usually a D2-B2-D-S-S) construct. See Figure 2 for an illustration of a simple, yet common type of argument.

The amount of diversity used is both limited and difficult to assess. In the severity branch only 3 out of 37 support patterns were diverse support. In the probability branch 17 out of 37 support patterns were diverse. However, only one of these occurrences was assessed as providing fully diverse support. The majority of the other diverse constructs were such that only one provided sufficient support,

while the other added some additional support (e.g. monitoring programmes in order to be able to respond quickly to incidents). Also, within the 17 occurrences of diverse support 10 were of the type “*P is y > x now, but will be reduced to x*”. We broke this down into two claims, each providing independently some support to the overall claim to preserve the structure of the narrative argument. It is conceivable that this could also be treated as a single support (see also the discussion for problems related to deriving a unique argument structure).

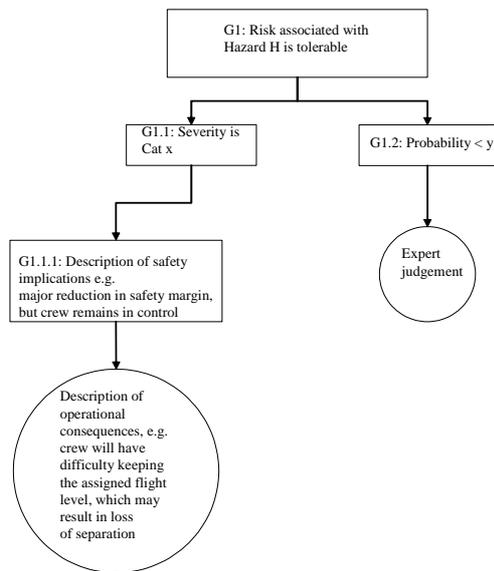


Figure 2: Example of a simple, but common argument structure.

This type of analysis reveals that the FHA consists predominantly of simple arguments at a very high level of abstraction. Many of the mitigating factors are not explained to a high level of detail. This makes a thorough analysis of potential dependencies or hidden assumptions in particular more difficult. A further issue complicating the analysis is the fact that the FHA was concerned specifically with RVSM mitigation. This means that the argument lacks details as far as other aspects are concerned. In practice these details would have increased the comprehensiveness of the argument.

4.3 Barrier analysis

A final stage in the analysis was to consider the use of barriers in the hazard mitigation arguments, see also (Smith et al., 2004) conducted a similar frequency assessment of medical and aviation related systems.

Overall, 21 preventive and 13 protective barriers were identified, see Table 2. The large majority of barriers (29 out of 34) were procedural, awareness-raising, or symbolic. As such they require interpretation by a human (e.g. application of CVSM, communication procedures, compulsory reporting points

etc). Only 5 barriers were of a different nature, e.g., the layout of the airspace, regulatory requirements, and monitoring programmes. However, it should be noted that apart from the layout of the airspace, the barriers in this class are to a certain extent organisational barriers equally relying on the interpretation and enactment of people.

Table 2: Types of barriers identified (the number in brackets refers to the number of different claims the barriers have been derived from).

	Pro-cedure	Train-ing / Awareness	Repres-entation / Symbolic	Organ-isation- al	Air-space	Σ
Pre-ventive	5 (2)	4	8 (2)	3	1	21
Protective	12	0	0	0	1	13
Σ	17	4	8	3	2	34

Given that air traffic control is such a technology-intensive domain it is surprising that there is little mention of any kind of technological barriers or technological support. As was mentioned in relation to the discussion of argument depth and breadth, there seems to be a tendency to regard barriers at high levels of abstraction, e.g. ‘compulsory reporting points at the entry and exit of RVSM space’, without explicit reference to how the barrier is realised and on what kind of support it relies.

This observation applies also to a number of assumptions made, for example ‘The crew will regain control’, without further specification how this would be achieved or what it is dependent on.

The way barriers are used (or left to be inferred) may be shaped by the type of argument that is constructed. The RVSM safety case argues that air traffic management will *remain* safe after *modification* to the *existing* air space. This is a special type of argument, which argues the safety of a new system by relying on an existing system and that system’s safety. In the case of the introduction of RVSM to the European air space the FHA does not make reference to or mention existing barriers. It does not provide a comprehensive account of how system safety is achieved but focuses on added features such as procedures that will be introduced with RVSM. This makes assessment more difficult because the dependence of certain barriers on other already existing barriers cannot be assessed.

The use of diverse barriers is based on breadth and diverse support patterns. There is only one instance of the use of diverse protective barriers, namely the introduction of compulsory reporting points, and the application of CVSM after a communication failure. Even here it may be assumed that the barriers are not independent, since the successful application of CVSM will depend on an approximate knowledge of the position of the aircraft, ensured by

the introduction of reporting points. There were two instances of fully diverse preventive barriers. In one case, however, the four diverse barriers were evaluated as having no impact on the risk classification. In the other case, the barrier function was to highlight the presence of a non-RVSM approved aircraft in RVSM space, which in turn is achieved by 7 or more (7 examples were given) diverse barrier systems, such as information on the radar, information on the flight strips etc.

5 CONCLUSION

This paper addresses two issues. Concepts and notions derived from considerations about the quality of arguments are investigated. These are interpreted in the particular case of hazard mitigation arguments. The paper explores how these concepts might facilitate the generation and representation of high-quality hazard mitigation arguments.

Hazard mitigation arguments appeal to barriers to demonstrate that hazards have been sufficiently controlled. This process typically brings together people from different backgrounds and of different areas of expertise. Arguments therefore integrate more or less successfully different conceptions of system safety. The paper also addresses the question of how the underlying model of system safety may be made explicit, so that it can be used subsequently in considerations of system safety, for example during the assessment of change implications.

Notions of depth and breadth of arguments are equated with the level of abstraction and the diversity of barriers that are appealed to in order to demonstrate that a hazard has been sufficiently controlled. The analysis of the RVSM FHA shows that the arguments have both limited depth and breadth. They represent high levels of abstraction and offer little diversity. However, depth and breadth are not to be adopted as measures of argument quality. There are various ways of measuring depth and breadth, and these may generate different values. In addition, a simple measure of depth and breadth by itself, does not offer any consistent insights. Arguments may be constructed in a number of different ways (in particular as far as depth is concerned), and the complexity of an argument does not necessarily correlate with the quality of an argument. The benefit of structural considerations is really that these concepts assist the developer of safety arguments in reasoning about the arguments, and they force people to consider explicitly the relationship of the various strands of the argument.

A small subset of hazards was analysed from one FHA and as such this does not present a representative picture. In addition, the amount of detail available was very low. As a consequence no meaningful representation of overall system safety can be de-

rived from the arguments provided in the functional hazard analysis. It was possible to deduce, however, that the number of diverse barriers used was low. From the analysis it can be concluded potentially that hazard mitigation arguments can provide a useful model of system safety. Making this model explicit could be used to assess dependencies, assumptions, and implications of future changes. The difficulty associated with this is a result of the complexity of the systems under consideration. For example, the argument for RVSM was concerned only with the effects introduced by this change. This argument would need to be integrated with all the other arguments, together making up an overall architecture of system safety. Therefore tools and mechanisms are required that can cope with this complexity and allow integration.

REFERENCES

- Bloomfield, R. & Littlewood, B. 2006. On the use of diverse arguments to increase confidence in dependability claims. In Besnard, D. et al. (Eds) *Structure for Dependability*. Springer Verlag . pp 254-268.
- Embrey, D. 1992. Quantitative and qualitative prediction of human error in safety assessments. Institute of Chemical Engineers Symposium Series 130, IChemE
- Eurocontrol. 2001a. EUR RVSM programme: Functional hazard assessment. Working Draft 1.0, European Organisation for the Safety of Air Navigation.
- Eurocontrol. 2001b. EUR RVSM programme: The EUR RVSM Pre-Implementation Safety Case. Version 2.0
- Eurocontrol. 2001c. Eurocontrol Safety Regulatory Requirement 4: Risk Assessment and Mitigation in ATM. Version 1.0
- Govier, T. 1988. *A practical study of arguments*. Wadsworth
- Hollnagel, E. 1999. Accidents and Barriers, In J-M Hoc et al. (Eds.) *Proceedings of Lex Valenciennes*, pp. 175-182. Volume 28, Presses Universitaires de Valenciennes.
- IEC, 1985. *Analysis techniques for system reliability: procedure for failure mode and effects analysis*
- Kelly, T. P. 1999. *Arguing Safety - A Systematic Approach to Managing Safety Cases*, PhD Thesis, Department of Computer Science, University of York, England.
- Kelly, T.P. & McDermid, J.A. 2001. *A Systematic Approach to Safety Case Maintenance, Reliability Engineering and System Safety*, volume 71, pp. 271-284, Elsevier
- Kletz, T. 1992. *Hazop and Hazan: Identifying and assessing process industrial hazards (3rd Ed.)*. IChemE
- Shorrock, S.T. & Kirwan, B. 2002 Development and application of a human error identification tool for air traffic control. *Applied Ergonomics* 33: 319-336.
- Smith, S.P., Harrison, M.D. & Schupp, B.A. 2004 How explicit are the barriers to failure in safety arguments?, In Heisel, M. et al. (Eds) *Proc SAFECOMP'04, Lecture Notes in Computer Science* Volume 3219 325-337, Springer.
- Sujan, M.A. et al. 2006. Qualitative analysis of dependability argument structure. In Besnard, D. et al. (Eds.) *Structure for dependability*, Springer Verlag pp.269-290.
- Toulmin, S.E. 1958. *The uses of arguments*, Cambridge University Press.
- UK Ministry of Defence 2004. Interim Def-Stan 00-56: *Safety Management Requirements for Defence Systems*.

Demonstration of Safety in Healthcare Organisations

Mark A. Sujan^{*}, Michael D. Harrison^{**},
Alison Steven⁺, Pauline H. Pearson⁺, Susan J. Vernon⁺⁺

^{*}Department of Computer Science
University of York, York, YO10 5DD, UK
sujan@cs.york.ac.uk

⁺School of Medical Education Development
University of Newcastle, Newcastle NE1
7RU, UK.
{alison.steven, p.h.pearson}@ncl.ac.uk

^{**}Informatics Research Institute,
University of Newcastle, Newcastle NE1
7RU, UK.
michael.harrison@ncl.ac.uk

⁺⁺School of Clinical Medical Sciences
University of Newcastle, Newcastle NE1
7RU, UK.
s.j.vernon@ncl.ac.uk

Abstract. The paper describes the current regulatory situation in England with respect to medical devices and healthcare providers. Trusts already produce evidence to the Healthcare Commission that they operate in accordance with standards set out by the Department of Health and the NHS. The paper illustrates how the adoption of an explicit goal-based argument could facilitate the identification and assessment of secondary implications of proposed changes. The NHS is undergoing major changes in accordance with its 10-year modernisation plan. These changes cannot be confined to the Trust level, but will have NHS-wide implications. The paper explores the possibility of an organisational safety case, which could be a useful tool in the management of such fundamental changes.

1. Introduction

Healthcare¹ organizations are undergoing major changes everywhere, both technical and organizational. The NHS in England is currently implementing a 10-year modernization plan [1] that will have implications for all areas of healthcare provision. Managing change in a safe and effective way poses major challenges. Similar restructurings of this scale have had serious implications, compare for example the privatisation and reorganisation of the UK railways. To deal with these implications in aviation, Eurocontrol explored the possibility of producing a whole-airspace safety case [2]. While there are substantial differences in the nature of these different domains, there are shared characteristics that lead to speculation about the

¹ All necessary clearances for the publication of this paper have been obtained. If accepted, the author will prepare the final manuscript in time for inclusion in the conference proceedings and will present the paper at the conference.

role of safety cases in healthcare. The shared characteristics include (with examples from healthcare):

- Integration of services and systems can mean that safety in one area is now more dependent on the behavior and performance of people and systems in other areas. This has occurred through the introduction of the Electronic Patient Record, through increased patient movements, and through the distributed location of patient data and of samples taken from the patient.
- Institutional and organisational changes can have many and far reaching consequences for system safety. This has occurred through new specialist roles mediating between primary and secondary care, and the trend to relocate large amounts of the budget to Primary Care Trusts.

This paper explores safety challenges within the health sector and how these might be addressed by supporting risk and change management through the construction and use of system-wide safety cases. The present exploration may be set within our broader work agenda. In particular there are a number of concerns:

- *Institutional and organizational issues (the scope of this paper):*
In this paper the institutional and regulatory background is discussed in order to identify, for example, relevant requirements, standards, stakeholders and the organisational structure. This is used to present the formal, structural and dynamic characteristics of a system-wide argument including appropriate level, ownership and structure.
- *Technical issues (future work):*
The discussion of the paper raises questions about how a safety case could be realised in practice. The last 10 years has seen substantial progress in safety case development. There has been a shift from prescriptive to goal-based regulation [3], and a graphical notation (GSN) has been developed [4], which facilitates the construction and communication of safety cases of large-scale systems. A study needs to investigate how this and subsequent work on maintenance [5] and on modularisation of safety cases [6] may render the construction of safety cases for healthcare organisations feasible.
- *Application of the safety case (future work):*
The activity of producing a safety case requires explicit consideration of safety-related issues, and provides assurance to both the organisation and to regulators that the system is adequately safe. The safety case has also the potential to be a useful tool in assessing the implications of change, both technical and organisational. A methodology for systematically utilising a system-wide safety case to support the management of change will be explored.

The next section describes the regulatory context in England. In this context two simplified yet realistic examples are considered. Firstly a technical change within a hospital environment is used to discuss how a goal-based argument could facilitate the assessment of implications of that change. Secondly an organisational change is considered in the context of the modernisation plan of the NHS in order to discuss the

possibility that a safety case could be used to support the management of this change. The concluding section discusses ongoing work and reflects on possible limitations.

2. The Regulatory Context in England

In England, as in other comparable European healthcare systems, there is a differentiation between manufacturers of medical devices on the one hand and healthcare providers as users or consumers of such devices on the other hand. Both are regulated by and are accountable to the Department of Health, albeit through different agencies and institutions. In general, manufacturers have to provide evidence that their devices are tolerably safe for a particular use in a specific environment. Healthcare providers, on the other hand, are audited to ensure that the care they provide meets national standards. A part of this is the requirement to utilise only previously certified medical devices.

The certification of medical devices within the UK environment

The UK Medical Devices Regulations 2002 (MDR 2002) implement a number of European directives relevant to the certification of medical devices:

1. Medical Devices Directive 93/42/EEC
2. IV-Diagnostic MDD 98/79/EC
3. Active Implantable MDD 90/385/EEC

The definition of what constitutes a medical device is broad and comprises devices as diverse as radiation therapy machines, syringes and wheelchairs. The Medicines and Healthcare Products Regulatory Agency (MHRA) acts as the *Competent Authority* overseeing the certification of medical devices. *Notified Bodies* of experts provide evaluation of high and medium risk medical devices undergoing certification to the Competent Authority.

The medical devices directive has three parts:

1. *Essential Requirements* that have to be met by any medical device to be marketed in the EU. Six requirements are regarded as essential including: defining acceptable levels of risk; applying safety principles during design and construction; establishing and meeting performance criteria, ensuring that undesirable side effects constitute an acceptable level of risk. Design and construction requirements are concerned with chemical, physical and biological properties, infection and contamination control, protection against radiation, protection against electrical, mechanical and thermal risk etc.

2. *Classification Rules* that specify four classes for medical devices. Class I devices pose little risk and are non-invasive. Classes IIa and IIb devices pose medium risk (medium to low risk, and medium to high risk, respectively), while Class III devices pose high risk.

3. *Conformity Routes* specifying different ways of manufacturer compliance with the Essential Requirements. In the case of Class I devices the manufacturer has to declare through a self-documentation process (no Notified Body is involved) that the Essential Requirements are met, and compile adequate technical documentation. For devices of the other classes a number of methods for demonstrating conformity are available. This is frequently done through a Full Quality Assurance System assessment, according to ISO13485:2003 (Quality Systems - Medical Devices: Particular Requirements for the Application of ISO9001). In the case of Classes IIa and IIb it is possible to provide evidence, including the results of risk analysis, test and inspection reports, design documentation, instructions for use and so on. The manufacturer is expected to have a systematic risk management process in place. ISO14971 (Medical Devices - Application of Risk Management to Medical Devices) is a harmonised standard specifying such a risk management process, focusing on risk analysis, risk evaluation, risk control and post-production information. Senior management is required to define acceptable levels of risk. Risk control through safety by design, mitigation measures, and appropriate warnings in the instructions for use must ensure that risks are reduced below a tolerable level and that residual risks do not exceed acceptable risk levels. Throughout the process a risk management file is maintained, which documents the activities of the risk management process.

All of these standards are addressed to the manufacturer of medical devices. When healthcare providers assemble different devices to create a system, the safety of the resulting system will not have been assured. As indicated in [7], the role of a systems integrator, with the responsibility of installing medical devices according to the manufacturers' instructions for use, of demonstrating the safety of the resulting system, and of providing documentation, training and support to the actual end users would be an important contribution to ensuring patient safety.

Apart from issuing instructions for use, the manufacturer has little influence on the way the devices are actually used in practice. More importantly, the manufacturer does not have detailed information about the specific environment and the processes within which the device will be operated within a particular healthcare provider's setting. It is reasonable to expect healthcare providers to demonstrate that the services they are providing are acceptably safe. Such a demonstration should make use of data supplied by the manufacturers.

Auditing of Healthcare Providers

Healthcare in England involves a diversity of actors. The Department of Health is responsible for setting the overall strategic direction of the NHS, for setting national standards for improving the quality of health services, and for securing adequate funding for the NHS. 28 Strategic Health Authorities (SHA) are responsible for setting and managing the local strategic direction of the NHS. The SHA develops plans to improve local services, and monitors the performance of healthcare providers within their region. The monitoring function is increasingly being taken over by the Healthcare Commission (HC), which assesses all healthcare providers against national standards. Primary Care Trusts (PCT) are local healthcare organisations

responsible for assessing the healthcare needs of the local communities, and for commissioning services from GPs, hospitals and so on. NHS Hospital Trusts manage hospitals ensuring healthcare provision is of sufficient quality, and that finances are managed effectively. PCTs purchase these services on behalf of their patients. On

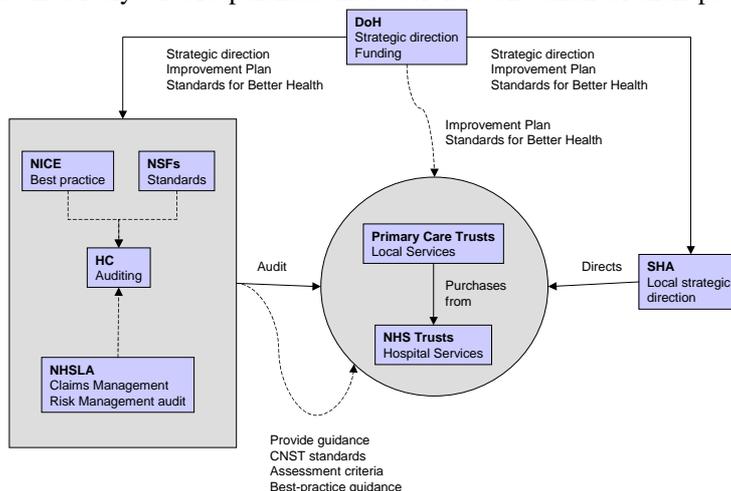


Figure 1: Simplified structure of regulatory context of healthcare provision in England

top of all this there are a large number of additional actors such as pharmacies, dentists, walk-in centres and so on. For the purpose of this paper it is sufficient to give a simplified version of the organisational structure including regulatory bodies and agencies, see fig. 1.

In 2004 the Department of Health published the Standards For Better Health [8] to set out quality expectations for all organisations providing NHS care in England. The standards focus on a broad spectrum of seven domains designed to cover the full spectrum of healthcare: safety; clinical and cost effectiveness; governance; patient focus; accessible and responsive care; care environment and amenities; public health. Each domain incorporates two types of standards: *core standards* and *developmental standards*. The 24 core standards are based on a number of standards or requirements that already exist. Developmental standards, on the other hand, outline requirements towards which continuous progress is expected.

Safety and risk management aspects are covered in particular in domains 1 (Safety) and 3 (Governance). For example, the Domain Outcome for *Safety* is specified as:

Patient Safety is enhanced by the use of health care processes, working practices and systemic activities that prevent or reduce the risk of harm to patients.

The corresponding core standards focus on learning from incident, fast response to incidents, adherence to NICE (National Institute for Clinical Excellence) guidance, decontamination of medical devices, minimisation of risks associated with the acquisition and use of medical devices etc. The developmental standard requires

healthcare providers to continuously review and improve all aspects of their activities that directly affect patient safety, and to apply best practice in assessing and managing risks to patients, staff and others.

The Healthcare Commission (HC) undertakes annual reviews of the provision of healthcare by each NHS organisation in England including PCTs, ambulance trusts, mental health trusts and acute trusts. These reviews aim to verify compliance with the core standards, as well as the achievement against the developmental standards. The HC has published guidance [9] that specifies the type of evidence to be produced in order to fulfil the standards set out by the Department of Health (e.g. participation in the National Reporting and Learning System).

The HC builds up a profile of information for every trust annually aimed at identifying trusts most at risk of not complying with the core standards, and areas where more thorough examination is required. The HC derives these profiles from its assessment and work programmes (e.g. improvement reviews, national staff surveys) as well as from other agencies and is active in seeking closer cooperation with bodies such as the NHS Litigation Authority (NHSLA), which audits risk management activities in healthcare organisations [10][11].

In conclusion therefore, within the regulatory context both manufacturers of medical devices and healthcare service providers are regulated and are required to provide evidence that their devices and services are tolerably safe and meet acceptable standards of quality. The producer - consumer relationship of manufacturers and healthcare has led to two regulatory contexts, which show little integration. Healthcare service providers are required to use only certified medical devices, and they have to react to patient safety alerts (with respect to medical devices) quickly, but there is no integration of assumptions and evidence produced by the manufacturers into a demonstration of safety produced by the healthcare organisation. The standards against which healthcare organisations are audited set out requirements that go beyond a given status-quo, and that emphasise continuous achievement and progress. As part of the requirement of Clinical Governance, healthcare providers are required to have a systematic risk management process in place. To demonstrate compliance with this requirement, healthcare providers produce prescribed evidence such as an official risk management strategy, including full allocation of responsibility and accountability, as well as evidence that the strategy is actually operational, such as minutes of risk management meetings. The regulator collects data throughout the year from a number of different sources. However, no formal argument (as required in aviation, for example) on the part of the healthcare organisation is required. This implies that assumptions and dependencies may not be documented properly, and that there are no formal notions of issues such as confidence in the evidence or diverse evidence to mitigate possible uncertainty.

3. Assessment of Technical Changes

This section describes a possible technical change in a hospital context. It is argued that a hospital-wide goal-based safety argument could facilitate the assessment of potentially adverse implications of the change and would enable an analysis of

dependencies that might otherwise go unnoticed.

Medication administration on a typical ward within a hospital relies on the nurse matching patient to the identity and quantity of drugs to be administered (see [12] for more details about the activities of the nurse and the checks performed). Patients and drugs are associated with one or more identifiers, including a unique case number, the patient’s name, the patient’s date of birth, and the drug name and dose. Since this practice has been established over many years it may be safely assumed that no risk assessment is or has been conducted. It may be assumed, however, that clinical risk management will aim to ensure that all incidents concerning patient mismatching are reported, and best-practice guidance issued by agencies such as the NPSA is implemented.

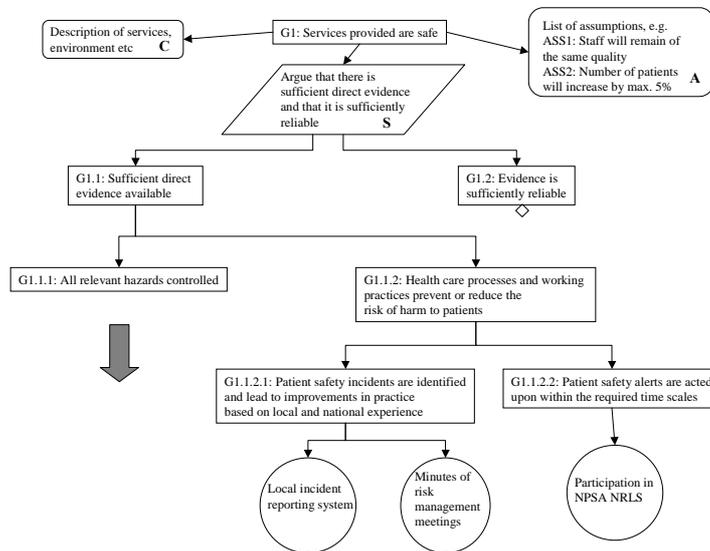


Figure 2: Simple top-level argument fragment

A formal hospital-wide safety case would include, as evidence, a risk assessment of the above activities that would support the claims that all hazards have been identified, risk assessed, and eliminated or reduced to be as low as reasonably practicable. This in turn could support the claim that all relevant hazards have been controlled. For the sake of illustration a simplistic top-level argument fragment is presented in fig. 2 and fig. 3.

The hazard mitigation argument forming part of the Functional Hazard Analysis (FHA) is depicted in fig. 4. Hazard mitigation arguments are usually described in narrative form. A graphical representation is chosen here to make the structure of the argument more clear. The argument is intended to demonstrate that the risk associated with wrong drug labels is tolerable. The argument relies on three key claims as well as one essential assumption:

- Claim G1.1.1: The most severe adverse events are caught in time, thus

- reducing overall severity
- Claim G1.2.2: The probability of wrong labels is than p_2
- Claim G1.2.1: The probability of the nurse not performing the cross-check is less than p_1
- The assumption (ASS1) is that there is always a nurse or a doctor close by and that they are attentive to changes in the symptoms of the patient.

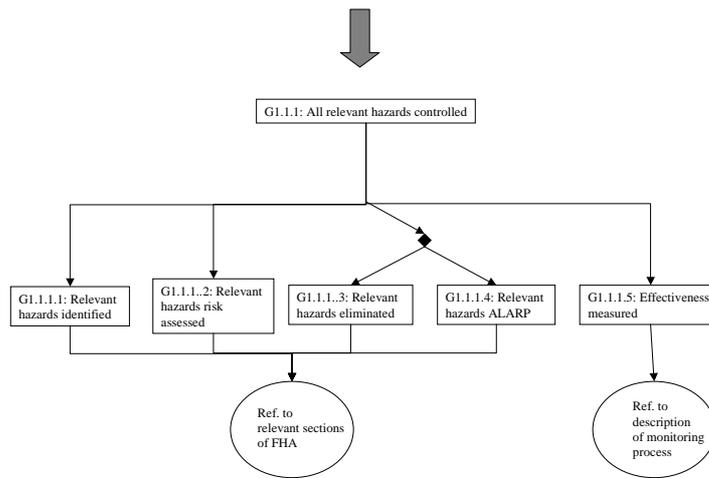


Figure 3: Simple top-level argument fragment (ctd.)

Consider a situation where the nurse (G1.2.1) and pharmacy (G1.2.1) claims are considered to be untenable and a technology solution, namely bar-coding, is proposed as a means of conforming to both targets. We can assume that because the bar-coding hardware and software is identified as a medical device the manufacturer will have conducted and documented a clinical trial and risk assessment of the device. Documentation associated with the device will include descriptions of the expected environment and procedures for use. Other assumptions will also have been made explicit, for example about maintenance and so on. The device would either be self-certified or certified by a Notified Body for marketing within the UK. The hospital as a consumer would purchase the device possibly with initial support for installation and operation. In purchasing the equipment the hospital would be required to conduct their own risk assessment. As a result of introducing the device it may be assumed that a number of activities will change substantially. The risk assessment would therefore identify and assess all changes to people directly involved, for example nurses and pharmacy staff. The corresponding hazard mitigation argument (if there was one) would modify the support for the claim G1.2.1 (nurse fails to perform cross-check). An assumption (ASS2) would now be required that the nurse follows the procedures (i.e., uses the bar-coding device to identify patients, and then administers drugs to the previously identified patient), and a claim that the bar-coding error rate is less than n (G1.2.1.2). An implicit assumption that the bar-coding hardware is available to the nurse would need to be made explicit, by claiming that the availability

of the hardware is $> m$ (G1.2.1.1). This claim would require further evidence that the hardware is regularly maintained (fig. 5).

To ensure that the availability claim is supported by valid evidence, sufficient technical staff time has to be allocated, training will have to be provided, facilities may have to be changed and so on. Establishing that this evidence is required may have the effect of altering the activities of the technicians (they may have to come to wards now when they previously did not), it requires their time, and it may conflict with similar statements made elsewhere for other hardware. Potential conflicts are of particular concern in the sense that they represent a class of assumptions about dependencies that often remain undetected.

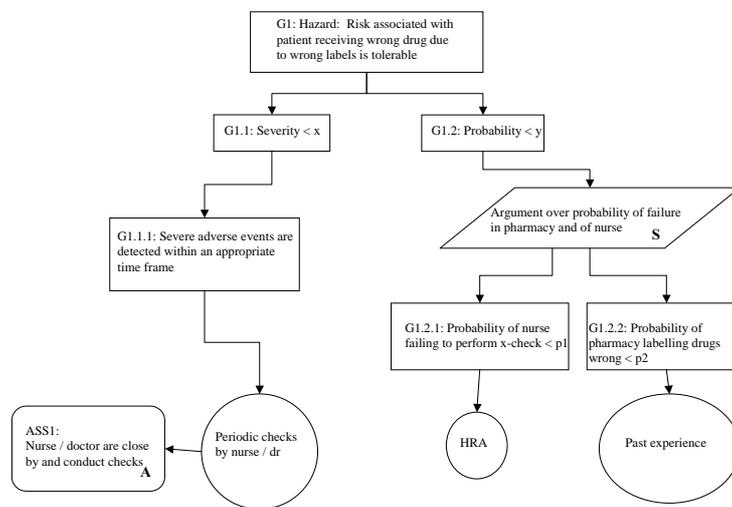


Figure 4: Hazard mitigation argument

Two problematic issues have been outlined through this scenario: the problem of the producer - consumer relationship of manufacturer and healthcare provider, and the problem associated with the tight integration of the healthcare organization. The first problem arises from the fact that medical devices and healthcare providers are certified and audited separately. Assumptions and information "hidden" within the documentation of the medical device may not be acknowledged or used properly within the auditing process of the healthcare provider. The tight integration of the various activities within the healthcare organization implies that changes cannot be assessed properly within their local context in relation to the activities that are immediately affected. Secondary effects may propagate throughout the organization. These effects may have unintended or adverse consequences if not properly taken into account. A (more) formal demonstration of safety in the form of an explicit argument could be a valuable tool. Such a demonstration would need to integrate information and assumptions of arguments produced by manufacturers into the larger perspective of an organization-wide safety argument. The process of producing a formal argument could in itself prove to be valuable, as it would prompt risk managers and

other stakeholders to reason about such issues. In addition, once such an organization-wide argument exists that makes all assumptions explicit, it will facilitate the assessment of both direct and secondary effects of changes, as well as previously hidden or undocumented dependencies. This could follow a systematic process similar to the one outlined in [5] for safety case maintenance.

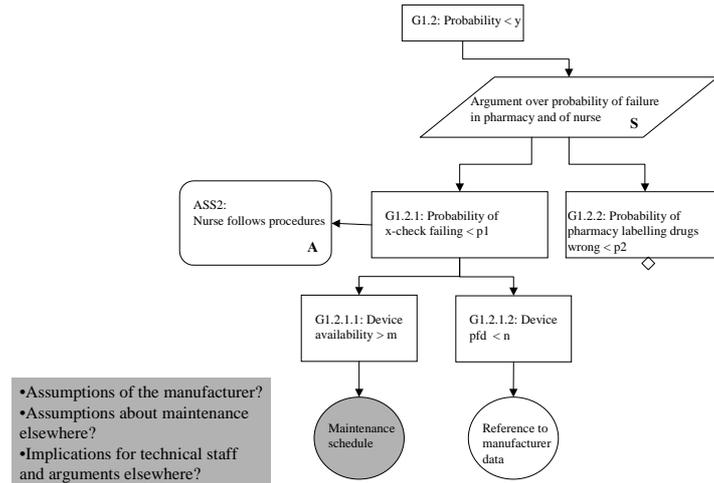


Figure 5: Hazard mitigation argument after introduction of bar-coding

4. Assessment of Organisational Changes

The successful management of organisational change is a major challenge in all industries. It can have far reaching consequences for the safety and the quality of the services provided, as well as for the service providers themselves. Modern organisations are complex systems characterised by emergent properties including those relating to safety. These properties arise from numerous, widespread and often unexpected interactions between internal and external actors and systems. Typically no formal representation justifying safe and efficient operation exists for such systems. Comprehensive representations of how the various actors and systems interact for the whole system are more or less completely absent even though limited safety cases of sub-systems sometimes do exist for the local environment. This makes the identification of interactions and dependencies accompanying changes difficult if not impossible. Prior assumptions made elsewhere and the complex propagation of secondary effects of change in other areas of the organisation are difficult to assess. We illustrate the effects of these changes by outlining a specific change scenario within the NHS that we will use to discuss some of the issues that may impact patient safety. The illustration will be used to motivate the formal and structural requirements of an organisation-wide safety case and how it might be used to manage organisational change.

Through the process of modernisation of the NHS [1][13] there is a transfer of responsibility to local authorities providing autonomous administration (via organizations such as the SHA and PCTs). The aim is that this change will enable service providers to respond flexibly and quickly to the different demands of their respective local communities. The rhetoric is that the service should transform itself from a sickness-service to a health-service, with stronger emphasis on prevention of disease and chronic disease management. As part of the personalisation of services, greater choice will be given to patients, and a stronger integration of local services will be achieved (e.g. mental health, social services). Overall this implies that all practitioners will increasingly be required to interact with teams of diverse characteristics. A further strand of the modernisation is the drive towards the utilisation of IT and of the internet. New services will be provided, for example electronic health spaces that contain the patient's medical history and data, as well as personalised information. Finally, there is a stronger focus on quality and safety of the services provided. This has already resulted in the establishment of new agencies such as NPSA and the HC, as well as in the provision of new standards such as the Standards for Better Health [8].

A major concern that has been a driver for many of these changes had been the problem of patient waiting times. Treatment waiting times of up to 18 months are to be reduced to a target of 18 weeks for the entire "patient journey" thereby leading to the patient perception that waiting times are no longer the main concern. One of the changes directed at achieving the aims of higher quality, personalised care, and reduced waiting times is the introduction of specialist nurses as mediators between primary and secondary care. A detailed description of the introduction of the role of a specialist nurse for urinary tract infections in children is given in [14]. In this paper we describe a simplified scenario in order to explore the implications of such change.

Urinary tract infections (UTI) in children may lead to renal scarring and other adverse consequences (hypertension, renal transplant etc) when not diagnosed and treated quickly and adequately. Previous published work has shown that GPs often lack a thorough understanding of proper management of such infections and of their possible consequences. The previous referral pathway for the investigation of childhood UTI required often a minimum of three interactions between practitioners and the children and their parents, taking up to a year. The patient presents to the GP, the GP may send a letter of referral to the hospital for further tests if there is any reason for suspecting an infection. In the hospital (some months later) the child and family are seen by a consultant paediatrician, a number of tests are organised and carried out at a later visit. Multiple tests may require multiple visits. A further visit to the hospital consultant or GP is required when the results are returned.

The aims of the introduction of a urinary tract infections specialist nurse included improved awareness among primary care teams, increased detection rates, reduction in time required of patients and parents, streamlined working with other agencies involved in the investigation process, reduction in overall duration of the process, and improved relationship with patients and parents.

The new solution is a nurse-led service for childhood UTI, combined with an education package for primary care teams, and available telephone support. The specialist nurse is autonomous, not supervised directly by a consultant. The nurse acts across the interface between primary care and secondary care. A consultant gets

involved only when the nurse determines that a particular case requires such attention based on clinical judgment. The primary care team member is required to complete a form that addresses a number of questions, and this form is passed to the nurse who assesses whether additional tests or consultant involvement are required. In cases where further tests are required, the nurse requests the tests and books the patients into the respective units in such a way that once the patient arrives in the hospital, all required tests have been organised for that day. In case of an abnormality of the test results, the respective unit can get in touch with the nurse directly. The nurse will make a decision based on the test result, and she will inform both the GP and the patient and parents promptly.

Proactive assessment of such change is not straightforward. The interface between primary and secondary care leads to a large number of interactions. The GP decision-making process is now different, from a diagnosis to a fast referral for in-depth consideration by the specialist nurse. This will release GP resource and will modify the GP's communication and relationship with hospital consultants. Testing services will interact only with the specialist nurse. The consultant will deal with a smaller proportion of cases. There will be a redistribution of resources to finance the new role for part of the Hospital Trust. The far-reaching consequences of this change (consequences that have only been hinted at) can only be assessed with proper models of the organisation, and in particular with models of how the organisation achieves safe operations. This becomes even more relevant when this change is seen in the broader view of all the other changes taking place concurrently. In addition to the UTI specialist nurse, it may be assumed that there are other similar specialist roles being introduced, all changing the activities of GPs, consultants, test facilities, patients and so on, and all possibly interacting with one another.

A formal safety argument could be used to make explicit how the overall organisation is achieving safe operations by making explicit the assumptions, dependencies and interactions that could be used to identify and to resolve interactions between changes. To make the use of such a safety argument possible a number of problems would need to be addressed. Currently, manufacturers of devices are certified by the Competent Authority. Healthcare providers (PCT, NHS Trust etc.), on the other hand, are audited independently by the HC. As already discussed changes often cut across sub-systems and responsibilities and may involve transfer of responsibility, transfer of resources and introduction of new technologies all at the same time. Any overarching safety argument would need to integrate information from all of these actors and, for this to happen, one actor would need to take overall responsibility. The SHA seems a possible candidate for this as it is involved in the process of performance monitoring of the healthcare providers, whereas auditing is increasingly being taken over by the HC. The SHA could thus assume the responsibility of compiling a safety argument for its area of responsibility. This safety argument would function as a tool within the management of change process rather than being part of the auditing process (as the HC does not audit the SHA) and would contribute to the achievement of the SHA's aim of providing higher-quality care within its region.

Achieving management of change through such arguments presents problems for the developers of safety arguments. They would need to integrate a substantial number of autonomous actors and would need to develop an underlying model upon

which the argument could be constructed. Traditional analytical models often fall short of providing adequate representations of organisations. A decomposition of the organisation into its elements for analytical reasons would not be an appropriate way of dealing with the emergent properties resulting from the manifold and complex interactions of all the elements of an organisation. Alternative models and representations are required that can interpret the organisation's defences or barriers in terms of human activity, and make this explicit to prevent unwanted and unsafe interactions.

5. Conclusion

This paper discusses the role that a safety argument might play in managing the safety (and other) implications of organizational change. Specifically, healthcare organizations are complex systems characterized by a large number of interactions and inter-relationships. Safety of such complex systems is an emergent property of these interactions. The complexity of healthcare organisations, the large number of autonomous actors, and the disjoint regulation of healthcare providers and medical device manufacturers renders the assessment of the implications of change very difficult.

The paper considers the introduction of technology in a hospital and demonstrates the problematic issues arising from producer-consumer relationship of manufacturer and healthcare provider, and from the tight integration of the healthcare organization.

We then considered the broader organisational issues associated with change and the role of whole-system arguments in managing organisational change. The very complexity of whole-system safety arguments makes the possibility of their construction and management a matter of concern. The example we used to illustrate this issue was relatively simple involving few organisations and yet many issues emerged through the discussion. For example no one actor in the organisation has responsibility for maintaining the whole safety of the system and therefore the overall safety argument. It is not clear how well existing argumentation techniques would manage the unforeseen emergent properties of these complex systems, nor is it clear how the overall high level complexity of the argument can be seen in relation to lower level issues, for example associated with the way that technologies like databases are used and shared and maintained safely across this complexity.

As yet no systematic techniques exist for managing effectively change through these arguments – an issue that we wish to explore in our future research agenda.

In terms of future agenda, we are particularly interested in exploring two models that might make it possible to relate barriers or defenses (and thus system safety) to the activities of the work environment and provide a means of managing the complexity. For example, consider the hazard arising from mixed-up drugs in the technological example of section 3. Barriers intended to prevent harm to the patient may include drug identity checks in the pharmacy before dispatch as well as drug identity checks at the bedside. As barriers do not simply exist as abstract entities, but are part of and realized through human activity, we aim to embed and to understand the operation of the barriers within the activities of the people carrying out the drug identity checks.

In the case of the nurse performing the final bedside check, for example, this barrier would be embedded in an activity of drug administration and the analysis would focus on the available tools and technologies (e.g. drug chart, labels, bar code system etc.) as well as the official procedures and less formal social rules. In addition, other activities, which may impact on the successful completion of this activity (and thus on the performance of the barrier) would be analyzed, e.g. the nurse's activity of providing assistance to other nurses (expressed in the division of labor within another activity). Expressing the barrier concept in such a way as meaningful, practical human activity provides a more meaningful model of the organization's emergent properties, and could thus be a useful underlying model for arguing safety of complex systems.

References

1. Department of Health: The NHS Plan: a plan for investment, a plan for reform (2000)
2. Kinnersly, S.: Whole Airspace ATM System Safety Case – Preliminary Study, Eurocontrol (2001)
3. Penny, J., Eaton, A., Bishop, P.G., Bloomfield, R.E.: The Practicalities of Goal-Based Safety Regulation, In Redmill, F. and Anderson, T. (eds): Proc. 9th Safety-Critical Systems Symposium, Springer Verlag (2001), pp. 35-48
4. Kelly, T.P.: Arguing Safety – A Systematic Approach to Safety Case Management, DPhil Thesis, Department of Computer Science, York (1998)
5. Kelly, T.P. and McDermid, J.A.: A Systematic Approach to Safety Case Maintenance, Reliability Engineering and System Safety 71, Elsevier (2001), pp. 271 – 284
6. Bate, I. and Kelly, T.: Architectural considerations in the certification of modular systems, Reliability Engineering and System Safety, 81, Elsevier (2003), pp. 303 – 324
7. Jordan, P.A.: Medical Device Manufacturers, Standards and the Law, paper presented at DIRC Workshop on Software Quality and the Legal System (2004)
8. Department of Health: Standards for Better Health (2004)
9. Healthcare Commission: Assessment for Improvement – the annual health check: Measuring what matters (2005)
10. NHS Litigation Authority: Risk Management Standard for Primary Care Trusts (2005)
11. NHS Litigation Authority: Clinical Negligence Scheme for Trusts General Clinical Risk Management Standards (2005)
12. Sujan, M.A., Henderson, J., Embrey, D.: Mismatching between planned and actual treatments in medicine – manual checking approaches to prevention, Human Reliability Associates (2004)
13. Department of Health: The NHS Improvement Plan: Putting people at the heart of public services (2004)
14. Coulthard, M.G., Vernon, S.J., Lambert, H.J., Matthews, J.N.S.: A nurse led education and direct access service for the management of urinary tract infections in children: prospective controlled trial, British Medical Journal (2003)

Part Eval – APPENDIX

(Methods and Tools for Resilience Evaluation)

Lessons Learned from the Deployment of a high-interaction Honeypot*

E. Alata¹, V. Nicomette¹, M. Kaâniche¹, M. Dacier², M. Herrb¹

¹LAAS-CNRS, ²Eurecom

Abstract

This paper aims at providing some precise information regarding the various steps that attackers go through in order to take control over a vulnerable machine. More importantly, it describes what can be learned from the observation of the attackers when logged on a compromised machine. The results are based on a six months period during which a controlled experiment has been run with a high interaction honeypot. We correlate our findings with those obtained with a worldwide distributed system of low-interaction honeypots. We provide precise information that can be used by those who are working, for instance, on intrusion detection systems or correlation engines. Indeed, detectors of various kinds make assumptions about the supposed behavior of attackers but few papers offer some concrete analysis of the characteristics of a large set of attacks of the same kind. This paper wishes to be one of the first of a kind in order to improve the global understanding of the threats against which we have to find efficient countermeasures.

1. Introduction

During the last decade, the Internet users have been facing a large variety of malicious threats and activities including viruses, worms, denial of service attacks, phishing attempts, etc. Several surveys and indicators, published at a regular basis, provide useful information about new vulnerabilities and security threats, with an indication of their estimated severities with respect to the potential damage that they might cause. On the other hand, several initiatives have been developed to monitor real world data related to malware and attacks propagation on the Internet. Among them, we can mention the Internet Motion Sensor project[4], CAIDA[16] and Dshield[9]. These projects provide valuable information for the identification and analysis of malicious activities on the Internet. Nevertheless, such information is not sufficient to model attack processes and analyze their impact on the security of the targeted machines. The CADHo project [2] in which we are involved is complementary to these initiatives and is aimed at filling such a gap by carrying out the following activities:

* A short version of this paper will appear in the Proceedings of EDCC-6, the 6th European Dependable Computing Conference, Coimbra, Portugal, October 18-20, 2006.

- deploying and sharing with the scientific community a distributed platform of honeypots[15] that gathers data suitable to analyze the attack processes targeting a large number of machines connected to the Internet;
- validating the usefulness of this platform by carrying out various analyses, based on the collected data, to characterize the observed attacks and model their impact on security.

A honeypot is a machine connected to a network but that no one is supposed to use. In theory, no connection to or from that machine should be observed. If a connection occurs, it must be, at best an accidental error or, more likely, an attempt to attack the machine. Two types of honeypots can be distinguished depending on the level of interactivity that they offer to the attackers. Low-interaction honeypots do not implement real functional services. They emulate simple services that cannot be compromised. Therefore, these machines cannot be used as stepping stones to carry out further attacks against third parties. On the other hand, high-interaction honeypots offer real services to the attackers to interact with which makes them more risky than low-interaction honeypots. As a matter a fact, they offer a more suitable environment to collect information on attackers activities once they manage to get the control of a target machine and try to progress in the intrusion process to get additional privileges. It is noteworthy that recently, hybrid honeypots combining the advantages of low and high-interaction honeypots have been also proposed [11, 6]. Both types of honeypots are investigated in the CADHo project to collect information about malicious activities on the Internet and to build models that can be used to characterize attackers behaviors and to support the definition and the validation of the fault assumptions considered in the design of secure and intrusion tolerant systems.

During the first stage of the project, we have focused on the deployment of a data collection environment (called Leurré.com[1]) based on low-interaction honeypots. As of today, around 40 honeypot platforms have been deployed at various sites from academia and industry in almost 30 different countries over the five continents. Each platform emulates three computers running Linux RedHat, Windows 98 and Windows NT, respectively, and various services such as `ftp`, `http`, etc. The data gathered by each platform are securely uploaded to a centralized database with the complete content, including payload of all packets sent to or from these honeypots, and additional information to facilitate its analysis, such as the IP geographical localization of packets' source addresses, the OS of the attacking machine, the local time of the source, etc.

Several analyses carried out on the data collected so far have revealed that very interesting conclusions can be derived with respect to the attack activities observed on the Internet [2, 12-15]. Nevertheless, with such honeypots, hackers can only scan ports and send requests to fake servers without ever succeeding in taking control over them. The second stage of the CADHo project is aimed at setting up and deploying high-interaction honeypots to allow us to analyse and model the behavior of malicious attackers once they have managed to compromise and get access to a new host, under strict control and monitoring. We are mainly interested in observing the progress of real attack processes and the activities carried out by the attackers in a controlled environment.

In this paper, we describe the preliminary lessons learned from the development and deployment of such a honeypot. It is important to stress the fact that the goal of this paper is not to present yet another architecture to build high interaction honeypots. Instead, it is about the rigorous definition of an environment to carry out a well thought off experiment aiming at better understanding and modeling the threats we are facing once an attacker has succeeded in taking over a machine connected to the Internet.

The main contributions of this paper are threefold. First, we do confirm the findings discussed in [14] showing that different sets of compromised machines are used to carry out the various stages of planned attacks. Second, we do outline the fact that, despite this apparent sophistication, the actors behind those actions do not seem to be extremely skillful, to say the least. Last, the geographical location of the machines involved in the last step of the attacks and the link with some phishing activities shed a geopolitical and socio-economical light on the results of our analysis.

The paper is organized as follows. Section 2 discusses some existing techniques for developing high-interaction honeypots and the design rationales for our solution. Section 3 describes our proposed solution. The lessons learned from the attacks observed over a period of almost 4.5 months are discussed in Section 4. Finally, Section 5 offers some conclusions as well as some ideas for future work.

2. Related work

The most obvious approach for setting up a high-interaction honeypot consists in the use of a physical machine and to dedicate it to record and monitor attackers activities. The installation of this machine is as easy as a normal machine. Nevertheless, probes must be added to store the activities. Operating in the kernel is by far the most frequent manner to do it. Sebek[7] and Uberlogger[3] operate in that way by using Linux Kernel Module (LKM) on Linux. More precisely, they launch a homemade module to intercept interesting system calls in order to capture the activities of attackers. Data collected in the kernel are stored on a server through the network. Communications with the server are hidden on all installed honeypots.

Another approach consists in using a virtual operating system [17]. User Mode Linux (UML) is a Linux compiled kernel which can be executed as other programs on a Linux operating system. It is possible to have several virtual operating systems running together on a single machine. Thanks to probes, activities on virtual operating systems are logged into the machine. In [18], an architecture of a honeypot using UML is presented. Uberlogger [3] can also be implemented in such an environment. VMware is also a tool used for emulation, but it emulates a whole machine instead of an operating system. Various operating systems (Windows, Gnu/Linux, etc) can be installed and executed together. It can also be used to deploy honeypots[19]. The problem with virtual honeypots is the possibility for the intruder to detect the presence of this virtual operating system [8]. Some well-known methods, available on Internet, allow the intruder to fingerprint VMware for example. Some solutions have been developed to hide the presence of VMware (see e.g. [10]).

Compared to honeypot-solutions based on physical machines, virtual honeypots provide a cost effective and flexible solution that is well suited for running experiments to observe attacks. In particular, the number of emulated systems and their configuration can be easily changed if needed.

3. Architecture of our honeypot

In our implementation, we have decided to use the VMware software and to install virtual operating system upon VMware. Our objective is to setup a high-interaction honeypot that can be easily configured and upgraded for different experimental studies. In particular, our intention is to emulate in the initial setup a limited number of machines, and then increase the number of emulated machines at a later stage of the project to have a more realistic target for attack that is representative of a real network.

As we explained in the Section 2, VMware workstation software [24] allows multiple operating systems to be run simultaneously on a single real host. With virtual operating systems, the cloning, the reconfiguration and the modification of the operating system are very simple. Furthermore, if the attacker succeeds in destroying some part of the operating system, the recovery procedure is simplified compared to the case of a real operating system.

In the following, we describe the configuration of the honeypot, and the data capture mechanisms.

3.1. Configuration

The objective of our experiment is to analyse the behavior of the attackers who succeed in breaking into a machine (a virtual host in our experiment). The vulnerability that he exploits is not as crucial as the activity he carries out once he has broken into the host. That's why we chose to use a simple vulnerability: weak passwords for `ssh` user accounts. In this way, our honeypot is not particularly hardened but this is intentional for two reasons. First, we are interested in analyzing the behavior of the attackers even when, once logged in, they exploit a buffer overflow and become root. So, if we use some kernel patch such as Pax [21] for instance, our system will be more secure but it will be impossible to observe some behavior. Secondly, if the system is too hardened, the intruders may suspect something abnormal and then give up.

In our setup, only `ssh` connections to the virtual host are authorized so that the attacker can exploit this vulnerability. A firewall blocks all connection attempts, but those to port 22 (`ssh`), from the Internet (see Figure 1).

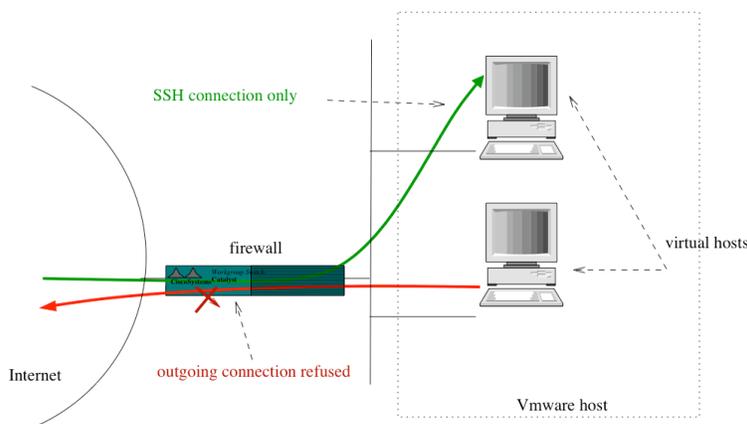


Figure 1- Topology of the honeypot

In order to prevent that intruders attack remote machines from the honeypot, a firewall blocks any connection from the virtual host to the outside. This does not prevent the intruder from downloading code, because he can use the `ssh` connection for that.¹

¹ As many intruders use outgoing `http` connections, we have sometimes authorized `http` connections in our experiments for a short time under our strict control by checking constantly that the attackers were not trying to attack other remote hosts.

This being said, as discussed later on, the lack of connectivity to the rest of the world by means of another protocol than `ssh` may look suspicious to a malicious user. We discuss the influence of this design choice in the section devoted to the analysis of the results of the experiments. We show that, instead of being a nuisance, it helps us discriminating between the various types of malicious users.

Our honeypot is a standard Gnu/Linux installation, with kernel 2.6, with the usual binary tools (compiler, usual commands, etc). No additional software was installed except the `httpd` apache server. This kernel was modified as explained in the next subsection. The real host is of course never used by regular users. The real operating system executing VMware is also a Gnu/Linux distribution with kernel 2.6.

3.2. Attackers activity logging

Our first objective is to log the activity of the intruders (the commands they use once they have broken into the honeypot) in a stealthy way. In order to log what the intruders do on the honeypot, we chose to modify some drivers functions `tty_read` and `tty_write` as well as the `exec` system call in the Linux kernel. The modifications of the functions `tty_read` and `tty_write` enable us to intercept the activity on all the terminals and pseudo-terminals of the system. The modification of the `exec` system call enables us to record the list of the system calls used by the intruder. These functions are modified in such a way that the captured information is logged directly into a buffer of the kernel memory. This means that the activity of the attacker is logged on the kernel memory of the honeypot itself. This approach is not common: in most of the approaches we have studied, the information collected is directly sent to a remote host through the network. The advantage of our approach is that logging into the kernel memory is difficult to detect by the intruder (more difficult at least than detecting a network connection).

The kernel memory of the virtual host is in fact an area of the whole memory of the real host. So, from the real host, an automatic script regularly inspects the area of the memory used by the virtual host, looking for a *magic word* indicating the beginning of the information logged into the kernel. This information is then recorded on the hard disk of the real host and then transferred to a database server (see Figure 2). This logging activity is executed on the real host, not on the virtual host, thus it is not easily detectable by the intruder (he cannot find anything suspicious in the list of the processes for example). Furthermore, we also decided to log this activity in such a way that it is not directly readable, even if the intruder succeeds in parsing the kernel memory of the honeypot. This activity information is compressed using the algorithm LZRW1[25] before being logged into the kernel memory. This makes it even more difficult for an intruder to detect that he is observed.

Our second objective is to record all the logins and passwords tried by the different attackers to break into the honeypot. For that purpose, we chose to add a new system call into the kernel of the virtual operating system and we have modified the source code of the `ssh` server so that it uses this new system call. The logins and passwords are then logged in the kernel memory, in the same buffer as the information related to the commands used by the attackers. As the whole buffer is regularly stored on the hard disk of the real host, we do not have to add other mechanisms to record these logins and passwords.

The activities of the intruder logged by the honeypot are preprocessed and then stored into an SQL database. The raw data are automatically processed to extract relevant information for further analyses, mainly: i) the IP address of the attacking machine, ii) the login and the password tested, iii) the date of the connection, iv) the terminal associated (`tty`) to each connection, and v) each command used by the attacker.

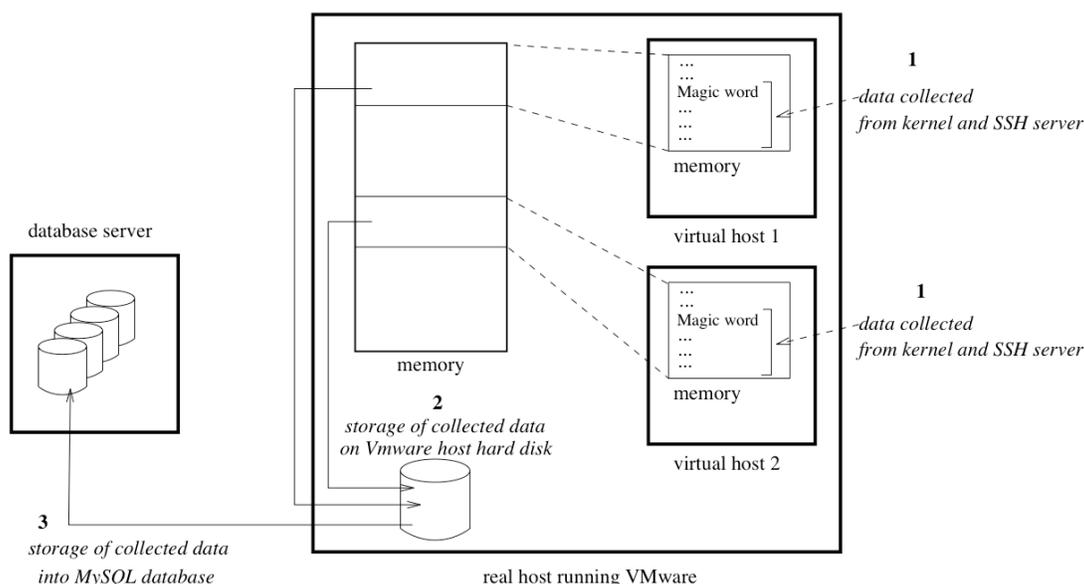


Figure 2- Data Collection

4. Experimental results

In this section, we present the results of our experiments. We give global statistics first, in order to give an overview of the activities observed on the honeypot, then we characterize the various intrusion processes. Finally, we analyse in detail the behavior of the attackers once they manage to break into the honeypot. In this paper, an *intrusion* corresponds to the activities carried out by an intruder who has succeeded to break into the system.

4.1. Global statistics

The high-interaction honeypot has been deployed on the Internet and has been running for 131 days during which 480 IP addresses have tried to contact its `ssh` port. It is worth comparing this value to the amount of hits observed against port 22, considering all the other low-interaction honeypot platforms we do have in the rest of the world (40 platforms). In the average, each platform has received hits on port 22 from around approximately 100 different IPs during the same period of time. Only four platforms have been contacted by more than 300 different IP addresses on that part and only one was hit by more visitors than our high interaction honeypot. Even better, the low-interaction platform maintained in the same subnet as the high-interaction studied in this paper experimented only 298 visits, i.e. less than two thirds of what the high-interaction did see. This very simple and first observation confirms the fact already described in [14] that some attacks are driven by the fact that attackers know in advance, thanks to scans done by other machines, where potentially vulnerable services are running. The existence of such a service on a machine will trigger more attacks against it. This is what we observe here: the low interaction machines do not have the `ssh` service open, as opposed to the high interaction one, and, therefore get less attacked than the one where some target has been identified.

The number of `ssh` connection attempts to the honeypot we have recorded is 248717 (we do not consider here the scans on the `ssh` port). This represents about 1900 connection attempts a day. Among these 248717 connection attempts, only 344 were successful. Table 1 represents the user accounts that were mostly tried (the top ten) as well as the total amount of different passwords that have been tested by the attackers. It is noteworthy that many user accounts corresponding to usual first names have also regularly been tested on our honeypot. The total number of accounts tested is 41530.

Before the real beginning of the experiment (approximately one and a half month), we had deployed a machine with a `ssh` server correctly configured, offering no weak account and password. We have taken advantage of this observation period to determine which accounts were mostly tried by automated scripts.

Account	Number of connection attempts	Percentage of connection attempts	Number of passwords tested
root	34251	13.77%	12027
admin	4007	1.61%	1425
test	3109	1.25%	561
user	1247	0.50%	267
guest	1128	0.45%	201
info	886	0.36%	203
mysql	870	0.35%	211
oracle	857	0.34%	226
postgres	834	0.33%	194
webmaster	728	0.29%	170

Table 1- `ssh` connection attempts and number of passwords tested

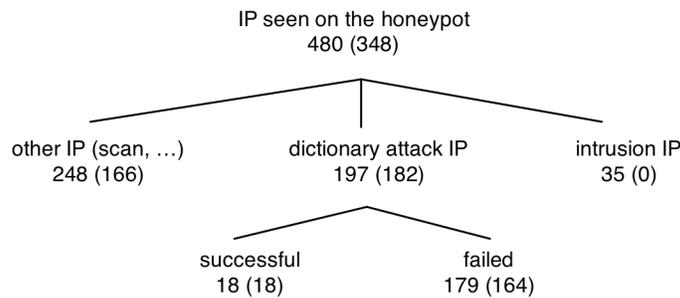
Using this acquired knowledge, we have created 17 user accounts and we have started looking for successful intrusions. Some of the created accounts were among the most attacked ones and others not. As we already explained in the paper, we have deliberately created user accounts with weak passwords (except for the `root` account). Then, we have measured the time between the creation of the account and the first successful connection to this account, then the duration between the first successful connection and the first real intrusion (as explained in section 4.2, the first successful connection is very seldom a real intrusion but rather an automatic script which tests passwords). Table 2 summarizes these durations (`UAi` means `User Account i`).

User Account	Duration between creation and first successful connection
UA1	1 day
UA2	Half a day
UA3	15 days
UA4	5 days
UA5	5 days
UA6	1 day
UA7	5 days
UA8	1 day
UA9	1 day
UA10	3 days
UA11	7 days
UA12	1 day
UA13	5 days
UA14	5 days
UA15	9 days
UA16	1 day
UA17	1 day

Table 2- Duration between the creation and the first successful connection for the broken accounts

4.2. Intrusion process

In the section, we present the conclusions of our analyses regarding the process to exploit the weak password vulnerability of our honeypot. The observed attack activities can be grouped into three main categories: 1) dictionary attacks, 2) interactive intrusions, 3) other activities such as scanning, etc.



X (Y): Y IP addresses among X were also seen on the low-interaction honeypots

Figure 3- Classification of observed IP addresses

As illustrated in figure 3, among the 480 IP addresses that were seen on the honeypot, 197 performed dictionary attacks and 35 performed real intrusions on the honeypot (see below for details). The 248 IP addresses left were used for scanning activity or activity that we did not clearly identified. Among the 197 IP addresses that made dictionary attacks, 18 succeeded in finding passwords. The others (179) did not find the passwords either because their dictionary did not include the accounts we created or because the corresponding weak password had already been changed by a previous intruder. We have also represented in Figure 3 the corresponding

number of IP addresses that were also seen on the low-interaction honeypot deployed in the context of the project in the same network (between brackets). Whereas most of the IP addresses seen on the high interaction honeypot are also observed on the low interaction honeypot, none of the 35 IPs used to really log into our machine to launch commands have ever been observed on any of the low interaction honeypots that we do control in the whole world ! This striking result is discussed here after.

4.2.1. Dictionary attack

The preliminary step of the intrusion consists in dictionary attacks². In general, it takes only a couple of days for newly created accounts to be compromised. As shown in Figure 3, these attacks have been launched from 197 IP addresses. By analysing more precisely the duration between the different `ssh` connection attempts from the same attacking machine, we can say that these dictionary attacks are executed by automatic scripts. As a matter of fact, we have noted that these attacking machines try several hundreds, even several thousands of accounts in a very short time.

We have made then further analyses regarding the machines that succeed in finding passwords, i.e., the 18 IP addresses. By searching the database containing information about the activities of these addresses against the other low interaction honeypots we found four important elements of information. First, we note that none of our low interaction honeypot has an `ssh` server running, none of them replies to requests sent to port 22. These machines are thus scanning machines without any prior knowledge on their open ports. Second, we found evidences that these IPs were scanning in a simple sequential way all addresses to be found in a block of addresses. Moreover, the comparison of the fingerprints left on our low interaction honeypots highlights the fact that these machines are running tools behaving the same way, not to say the same tool. Third, these machines are only interested in port 22, they have never been seen connecting to other ports. Fourth, there is no apparent correlation as far as their geographical location is concerned: they are located all over the world.

In other words, it comes from this analysis that these IPs are used to run a well known program. The detailed analysis of this specific tool lies outside the scope of the paper but, nevertheless, it is worth mentioning that the activities linked to that tool, as observed in our database thanks to all our platforms, indicate that it is unlikely to be a worm but rather an easy to use and widely spread tool.

4.2.2. Interactive attack: intrusion

The second step of the attack consists in the real intrusion. We have noted that, several days after the guessing of a weak password, an interactive `ssh` connection is executed on our honeypot to issue several commands. We have reason to believe that, in those situations, a real human being, as opposed to an automated script, is connected to our machine. This is explained and justified in Section 4.3. As shown in Figure 3, these intrusions come from 35 IP addresses never observed on any of the low-interaction honeypots.

Whereas the geographic localisation of the machines performing dictionary attacks is very blur, the machines that are used by a human being for the interactive `ssh` connection are, most of the time, clearly identified. We

² Let us note here that we consider as “dictionary attack” any attack that tries more than 10 different accounts and passwords.

have a precise idea of their country, geographic address, the responsible of the corresponding domain. Surprisingly, these machines, for half of them, come from the same country, an European country not usually seen as one of the most attacking ones as reported, for instance, by the www.leurrecom.org web site.

We then made analyses in order to see if these IP addresses had tried to connect to other ports of our honeypot except for these interactive connections; and the answer is no. Furthermore, the machines that make interactive `ssh` connections on our honeypot do not make any other kind of connections on this honeypot, i.e. no scan or dictionary attack. Further analyses, using the data collected from the low-interaction honeypots deployed in the CADHo project, revealed that none of the 35 IP addresses have ever been observed on any of our platforms deployed in the word. This is interesting because it shows that these machines are totally dedicated to this kind of attack (they only targeted our high-interaction honeypot and only when they knew at least one login and password on this machine).

We can conclude for these analyses that we face two groups of attacking machines. The first group is composed of machines that are specifically in charge of making dictionary attacks. Then the results of these dictionary attacks are published somewhere. Then, another group of machines, which has no intersection with the first group, comes to exploit the weak passwords discovered by the first group. This second group of machines is, as far as we can see, clearly geographically identified and commands are executed by a human being. A similar two steps process was already observed in the CADHo project when analyzing the data collected from the low-interaction honeypots (see [14] for more details).

4.3. Behavior of the attackers

This section is dedicated to the analysis of the behavior of the intruders. We first characterize the intruders, i.e. we try to know if they are humans or programs. Then, we present in more details the various actions they have carried out on the honeypot. Finally, we try to figure out what their skill level seems to be.

- We concentrate the analyses on the last three months of our experiment. During this period, some intruders have visited our honeypot only once, others have visited it several times, for a total of 38 `ssh` intrusions. These intrusions were initiated from 16 IP addresses and 7 accounts were used. Table 3 presents the number of intrusions per account, IP addresses and passwords used for these intrusions.

Account	Number of intrusions	Number of passwords	Number of IP addresses
UA2	1	1	1
UA4	13	2	2
UA5	1	1	1
UA8	1	1	1
UA10	9	2	2
UA13	6	1	5
UA16	5	1	3
UA17	2	1	1

Table 3- Number of intrusions per account

It is of course very difficult to be sure that all the intrusions for a same account are initiated by the same person. Nevertheless, in our case, we noted that:

- most of the time, after his first login, the attacker changes the weak password into a strong which, from there on, remains unchanged.
- when two different IP addresses access the same account (with the same password), they are very close and belong to the same country or company.

4.3.1. Types of the attackers: human or programs

Before analyzing what intruders do when connected, we can try to identify who they are. They can be of two different natures. Either they are humans, or they are programs which reproduce simple behaviors. For all intrusions but 12, intruders have made mistakes when typing commands. Mistakes are identified when the intruder uses the backspace to erase a previously entered character. So, it is very likely that such activities are carried out by a human, rather than programs.

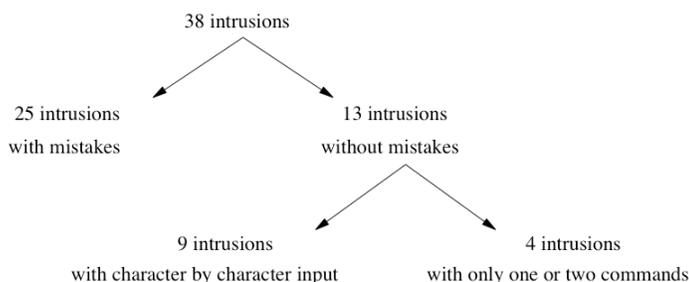


Figure 4- Characterization of the intrusions

When an intruder did not make any mistake, we analyse how the data are transmitted from the attacker machine to the honeypot. We can note that, for `ssh` communications, data transmission between the client and the server is asynchronous. Most of the time, the `ssh` client implementation uses the function `select()` to get user input. So, when the user presses a key, this function ends and the program sends the corresponding value to the server. In the case of a copy and a paste into the terminal running the client, the `select()` function also ends, but the program sends all the values contained in the buffer used for the paste into the server. We can assume that, when the function `tty_read()` returns more than one character, these values have been sent after a copy and a paste. If all the activities during a connection are due to a copy and a paste, we can strongly assume that it is due to an automatic script. Otherwise, this is quite likely a human being who uses shortcuts from time to time (such as CTRL-V to paste commands into its `ssh` session). For 7 out of the last 12 activities without mistakes, intruders have entered several commands on a character by character basis. This, once again, seems to indicate that a human being is entering the commands. For the 5 others, their activities are not significant enough to conclude: they have only launched a single command, like `w`, which is not long enough to highlight a copy and a paste.

4.3.2 Attackers activities

The first significant remark is that all of the intruders change the password of the hacked account. The second remark is that most of them start by downloading some files. In all, but one, cases the attackers have tried to

download some malware to the compromised machines. In a single case, the attacker has first tried to download an innocuous, yet large, file to the machine (the binary for a driver coming from a known web site). This is probably a simple way to assess the quality of the connectivity of the compromised host.

The command used by the intruders to download the software is `wget`. To be more precise, 21 intrusions upon 38 include the `wget` command. These 21 intrusions concern all the hacked accounts. As mentioned in section 3.1, outgoing `http` connections are forbidden by the firewall. Nevertheless, the intruders still have the possibility to download files through the `ssh` connection using `sftp` command (instead of `wget`). Thus, it is interesting to analyse the percentage of the attackers that continue their attack despite this `wget` problem. Surprisingly, we noted that only 30% of the intruders did use this `ssh` connection. 70% of the attackers were unable to download their malware due to the absence of `http` connectivity! Three explanations can be envisaged at this stage. First, they follow some simplistic cookbook and do not even know the other methods at their disposal to upload a file. Second, the machines where the malware resides do not support `sftp`. Third, the lack of `http` connectivity made the attacker suspicious and he decided to leave our system. Surprisingly enough, the first explanation seems to be the right one in our case as we observe them leaving the machine after an unsuccessful `wget` and coming back a few hours or days later, trying the same command again as if they were hoping it to work at that time. Some of them have been seen trying this several times. It comes out of this that i) they are apparently unable to understand why the command fails, ii) they are not afraid to come back to the machine despite the lack of `http` connectivity, iii) applying such brute force attack reveals that they are not aware of any other method to upload the file.

Once they manage to download their malware using `sftp`, they try to install it (by decompressing or extracting files for example). 75% of the intrusions that installed software did not install it on the hacked account but rather on standard directories such as `/tmp`, `/var/tmp` or `/dev/shm` (which are directories with write access for everybody). This makes the activity of the hacker more difficult to identify because these directories are regularly used by the operating system itself and shared by all the users.

Additionally, we have identified four main activities of the intruders. The first one is launching `ssh` scans on other networks but these scans have never tested local machines. Their idea is to use the targeted machine to scan other networks, so that it is more difficult for the administrator of the targeted network to localize them. The program used by most intruders, which is easy to find on the Internet, is `pscan.c`.

The second type of activity consists in launching `irc` clients, e.g., `emech` [20] and `psyBNC`. Names of binary files have regularly been changed by intruders, probably in order to dissimulate them. For example, the binary files of `emech` have been changed to `crond` or `inetd`, which are well known binary file names and processes on Unix systems.

The third type of activity is trying to become root. Surprisingly, such attempts have been observed for 3 intrusions only. Two rootkits were used. The first one exploits two vulnerabilities: a vulnerability which concerns the Linux kernel memory management code of the `mremap` system call [23] and a vulnerability which concerns the internal kernel function used to manage process's memory heap [22]. This exploit could not succeed because the kernel version of our honeypot does not correspond to the version of the exploit. The intruder should have realized this because he checked the version of the kernel of the honeypot (`uname -a`). However, he launched this rootkit anyway and failed. The other rootkit used by intruders exploits a vulnerability in the program `ld`. Thanks to this exploit, three intruders became `root` but the buffer overflow succeeded only

partially. Even if they apparently became `root`, they could not launch all desired programs (removing files for example caused access control errors).

The last activity observed in the honeypot is related to phishing activities. It is difficult to make precise conclusions because only one intruder has attempted to launch such an attack. He downloaded a forged email and tried to send it through the local `smtp` agent. But, as far as we could understand, it looked like a preliminary step of the attack because the list of recipient emails was very short. It seems that it was just a preliminary test before the real deployment of the attack.

4.3.3. Attackers skill

Intruders can roughly speaking be classified into two main categories. The most important one is relative to *script kiddies*. They are inexperienced *hackers* who use programs found on the Internet without really understanding how they work. The next category represents intruders who are more dangerous. They are named "black hat". They can make serious damage on systems because they are expert in security and they know how to exploit vulnerabilities on various systems.

As already presented in §4.3.2. (use of `wget` and `sftp`), we have observed that intruders are not as clever as expected. For example, for two hacked accounts, the intruders don't seem to really understand the Unix file access rights (it's very obvious for example when they try to erase some files whereas they don't have the required privileges). For these two same accounts, the intruders also try to kill the processes of other users. Many intruders do not try to delete the file containing the history of their commands or do not try to deactivate this history function (this file depends on the login shell used, it is `.bash_history` for example for the `bash`). Among the 38 intrusions, only 14 were cleaned by the intruders (11 have deactivated the history function and 3 have deleted the `.bash_history` file). This means that 24 intrusions left behind them a perfectly readable summary of their activity within the honeypot.

The IP address of the honeypot is private and we have started another honeypot on this network. This second honeypot is not directly accessible from the outside, it is only accessible from the first honeypot. We have modified the `/etc/motd` file of the first honeypot (which is automatically printed on the screen during the login process) and added the following message: "In order to use the software XXX, please connect to A.B.C.D". In spite of this message, only one intruder has tried to connect to the second honeypot. We could expect that an experienced hacker will try to use this information. In a more general way, we have very seldom seen an intruder looking for other active machines on the same network.

One important thing to note is relative to fingerprinting activity. No intruder has tried to check the presence of VMware software. For three hacked accounts, the intruders have read the contents of the file `/proc/cpuinfo` but that's all. None of the methods discussed on Internet was tested to identify the presence of VMware software [8][5]. This probably means that the intruders are not experienced hackers.

Conclusion

In this paper, we have presented the results of an experiment carried out over a period of 6 months during which we have observed the various steps that lead an attacker to successfully break into a vulnerable machine and his behavior once he has managed to take control over the machine.

The findings are somehow consistent with the informal know how shared by security experts. The contributions of the paper reside in performing an experiment and rigorous analyses that confirm some of these informal assumptions. Also, the precise analysis of the observed attacks reveals several interesting facts. First of all, the complementarity between high and low interaction honeypots is highlighted as some explanations can be found by combining information coming from both set ups. Second, it appears that most of the observed attacks against port 22 were only partially automatised and carried out by script kiddies. This is very different from what can be observed against other ports, such as 445, 139 and others, where worms have been designed to completely carry out the tasks required for the infection and propagation. Last but not least, honeypot fingerprinting does not seem to be a high priority for attackers as none of them has tried the known techniques to check if they were under observation. It is also worth mentioning a couple of important missing observations. First, we did not observe scanners detecting the presence of the open ssh port and providing this information to other machines in charge of running the dictionary attack. This is different from previous observations reported in [14]. Second, as most of the attacks follow very simple and repetitive patterns, we did not observe anything that could be used to derive sophisticated scenarios of attacks that could be analysed by intrusion detection correlation engine. Of course, at this stage it is too early to derive definite conclusions from this observation.

Therefore, it would be interesting to keep doing this experiment over a longer period of time to see if things do change, for instance if a more efficient automation takes place. We would have to solve the problems of weak passwords being replaced by strong ones though, in order to see more people succeeding in breaking into the system. Also, it would be worth running the same experiment by opening another vulnerability into the system and verifying if the identified steps remain the same, if the types of attackers are similar. Could it be, at the contrary, that some ports are preferably chosen by script kiddies while others are reserved to some more elite attackers? This is something that we are in the process of assessing.

Acknowledgement. This work has been partially supported by: 1) CADHo, a research action funded by the French ACI “Sécurité & Informatique” (www.cadho.org), 2) The CRUTIAL European project IST-027513 (crutial.cesiricerca.it), and 3) the ReSIST European Network of Excellence IST-026764 (www.resist-noe.org).

References

- [1] Home page of Leurre.com: <http://www.leurre.org>.
- [2] E. Alata, M. Dacier, Y. Deswarte, M. Kaaniche, K. Kortchinsky, V. Nicomette, Van. Hau Pham, and F. Pouget, Collection and analysis of attack data based on honeypots deployed on the Internet. Proc. of QOP 2005, 1st Workshop on Quality of Protection (co-located with ESORICS and METRICS), Sept. 15, Milan, Italy, 2005.
- [3] J. Babès, I. Alberdi and E. Le Jamtel, Uberlogger : un observatoire niveau noyau pour la lutte informatique défensive. Proc. of SSTIC '05, Symposium sur la Sécurité des Technologies de l'Information et des Communications, Rennes, France, 2005.
- [4] M. Bailey, E. Cooke, F. Jahanian, and J. Nazario, The Internet motion sensor - a distributed blackhole monitoring system. In Proc. Network and Distributed Systems Security Symp. (NDSS 2005), San Diego, USA, 2005.
- [5] J. Corey, Advanced honeypot identification and exploitation. Phrack, N 63, Available on: <http://www.phrack.org/fakes/p63/p63-0x09.txt>.

- [6] W. Cui, V. Paxson, N. Weaver, and R. H. Katz, Protocol-independent adaptive replay of application dialog. In Proceedings of NDSS 2006, Network and Distributed Systems Security Symposium, 2006.
- [7] M. Davis, E. Balas, J. de Haas. Available on: <http://www.honeynet.org/tools/sebek>.
- [8] T. Holz and F. Raynal, Detecting honeypots and other suspicious environments. In Systems, Man and Cybernetics (SMC) Information Assurance Workshop. Proc. from the Sixth Annual IEEE, pages 29--36, 2005.
- [9] <http://www.dshield.org>. Home page of the DShield.org Distributed Intrusion Detection System, <http://www.honeynet.org>.
- [10] K. Kortchinsky. Patch for vmware. Available on: <http://honeynet.rstack.org/tools/vmpatch.c>.
- [11] C. Leita, K. Mermoud, and M. Dacier. Scriptgen: an automated script generation tool for honeyd. In Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC 2005), 2005.
- [12] Project Leurre.com. Publications web page: <http://www.leurrecom.fr/paper.htm>.
- [13] M. Dacier, F. Pouget, and H. Debar. Honeypots: practical means to validate malicious fault assumptions. In Proc. 10th IEEE Pacific Rim Int. Symp., pages 383--388, Tahiti, French Polynesia, 2004.
- [14] F. Pouget, M. Dacier, and V. Hau Pham. Understanding threats: a prerequisite to enhance survivability of computing systems. In Proc. IISW'04, Int. Infrastructure Survivability Workshop 2004 (in conjunction with the 25th IEEE Int. Real-Time Systems Symp. (RTSS 04), Lisboa, Portugal, 2004.
- [15] F. Pouget, M. Dacier, and V. Hau Pham. Leurre.com: on the advantages of deploying a large scale distributed honeypot platform. In Proc. of ECCE'05, E-Crime and Computer Conference, Monaco, 2005.
- [16] CAIDA Project. Home Page of the CAIDA Project, <http://www.caida.org>.
- [17] Honeynet Project. Know your enemy: Defining virtual honeynets. Available on: <http://www.honeynet.org>.
- [18] Honeynet Project. Know your enemy: Learning with user-mode linux. Available on: <http://www.honeynet.org>.
- [19] Honeynet Project. Know your enemy: Learning with vmware. <http://www.honeynet.org>.
- [20] EnergyMech team. Energymech. Available on: <http://www.energymech.net>.
- [21] The PaX Team. Available on: <http://pax.grsecurity.net>.
- [22] US-CERT. Linux kernel do_brk() function contains integer overflow. Available on: <http://www.kb.cert.org/vuls/id/981222>.
- [23] US-CERT. Linux kernel mremap(2) system call does not properly check return value from do_munmap() function. Available on: <http://www.kb.cert.org/vuls/id/981222>.
- [24] Inc. VMware. Available on: <http://www.vmware.com>.
- [25] R. N. Williams, An extremely fast ziv-lempel data compression algorithm. In Data Compression Conference, pages 362--371, 1991.

Robustness of the Device Driver-Kernel Interface: Application to the Linux Kernel*

Arnaud Albinet, Jean Arlat, Jean-Charles Fabre

LAAS-CNRS

Abstract

Device driver programs compose the larger part of operating systems. Previous studies have shown that such kernel extensions contribute the most to the sources of operating system misbehavior. Their failure can have significant impact on the kernel and cause significant damages to the system as a whole. This chapter aims at assess objectively and efficiently the robustness of an operating system in the presence of faulty drivers. Towards this ends we propose to conduct fault injection experiments targeting the DPI (Driver Programming Interface) that implements the way driver programs interact with the kernel. Faults are injected on the parameters of these kernel core functions. This allows for the derivation of useful results about the failure modes induced and thus on the characterization of the robustness of a target kernel with respect to faulty drivers. To conduct comprehensive analyses, complementary benchmarking measures are considered that span three viewpoints: kernel responsiveness, kernel availability and workload safety. The experimental data gathered can also help isolate weaknesses and reveal potential error propagation channels; such insights might be useful to derive protection mechanisms focusing on identified malfunctions. The various forms of awareness brought in by these results are useful for a large set of end-users: system user, system integrator and operating system developer.

1. Introduction

Dependability concerns, encompassing robustness assessment, is an essential question before a developer can make the decision whether to integrate off-the-shelf (OTS) components into a dependable system. Here, and in what follows, robustness is understood as the degree to which a system operates correctly in the presence of exceptional inputs or stressful environmental conditions, in compliance with the generic definition (*dependability with respect to external faults*) given in [Avižienis *et al.* 2004].

From cost-effectiveness viewpoint, operating systems and kernels are privileged OTS components as candidates for integration into a system. However, integrators are often reluctant to make such a move without obtaining a deeper knowledge and understanding about such a component beyond functional issues, in particular with respect to its failure modes and its behavior in presence of faults. Due to the opacity that is often attached to the commercial offer and to the difficulty and significant cost associated to the availability of the source code, the Open Source option, for which access to the source code is granted, is progressively making its way as an attractive and promising alternative. Also, results of many studies have shown that Open Source solutions did

* To appear in: K. Kanoun and L. Spainhower, Eds., *Dependability Benchmarking*, IEEE CS Press, 2007.

not exhibit significantly more critical failure modes and in some cases they were even found to demonstrate superior behaviors than commercial options [Koopman & DeVale 1999, Arlat *et al.* 2002, Marsden *et al.* 2002, Vieira & Madeira 2003]. In this chapter, we will simply denote such components (either Commercial or Open-Source), as OTS components.

In the past years, several experimental studies have addressed this important issue from different perspectives [Koopman & DeVale 1999, Arlat *et al.* 2002, Madeira *et al.* 2002]. This has also led to the proposal of tentative dependability benchmarking approaches, aimed at characterizing the robustness of computer systems and OTS [Tsai *et al.* 1996, Mukherjee & Siewiorek 1997, Brown & Patterson 2000, Zhu *et al.* 2003]. However, such proposals are still preliminary, and they did not reach yet the level of recognition attached to performance benchmarks. The DBench project, in which this work was included, was another major contribution aimed at promoting such a kind of approach by defining a comprehensive framework for the definition and implementation of dependability benchmarks [Kanoun *et al.* 2002, Kanoun *et al.* 2005a].

A large part of the code that makes up an operating system consists of device driver programs. For example, in the case of *Linux*, drivers have consistently represented more than half of the source code [Godfrey & Tu 2000]. This ratio is smoothly increasing: recent releases account for about 60 percent of the code [Gu *et al.* 2003]. More importantly, as the whole size of the kernel is rapidly growing, this results in an exponential increase of the number of lines of code of the driver programs. Such programs are commonly developed by third-party hardware device experts and integrated by kernel developers. This process is not always well mastered and an erroneous behavior of such programs that are intimately connected to the kernel may have dramatic effects. As pointed out in [Murphy & Levidow 2000] for *Windows* and as shown by the analysis of the *Linux* source code carried out in [Chou *et al.* 2001], a significant proportion of operating system failures can be traced to faulty drivers. Things are not improving much: indeed, as quoted in [Swift *et al.* 2004], in *Windows XP*, driver programs are reported to account for 85% of recently reported crash failures.

It is thus necessary to investigate and propose new methods, beyond the collection of field data, for specifically analyzing the impact of faulty drivers on operating systems. Fault injection techniques, where faulty behaviors are deliberately provoked to simulate the activation of faults provide a pragmatic and well-suited approach to support such an analysis. Among the fault injection techniques, the software-implemented fault injection (SWIFI) technique (e.g., see [Carreira *et al.* 1998]) provides the proper level of flexibility and low intrusiveness to address this task. Based on these principles, we have developed an experimental environment for the evaluation of the robustness of the *Linux* kernel when faced to abnormal behaviors of its driver programs.

To our knowledge, very few research studies have been reported on this topic. The work reported in [Edwards & Matassa 2002] concerns also the *Linux* kernel, but focuses rather on the dual problem of characterizing the robustness of driver programs when subjected to hardware failures. The authors have devised a sophisticated approach to inject faults in the driver under test that is relying on the appealing notion of Common Driver Interface (CDI) that specifies the driver interactions within the kernel space. In [Gu *et al.* 2003], the authors have conducted a comprehensive dependability analysis of the *Linux* kernel. However, in this study fault injection has been related to the execution stream of the kernel code: more precisely, a selected set of functions of the kernel has been targeted: namely, the processor dependent code, the file system support code, the core kernel code, and the memory management code. Our concern is rather the analysis of the robustness of an operating system kernel in presence of faulty drivers. In line with this objective, but considering several instances of the *Windows* series, in [Durães & Madeira 2003] the authors have used mutations of the executable code of the driver to simulate a faulty driver.

The work reported in this chapter is rather complementary, in the sense that we investigate an alternative approach: fault injection is targeting the parameters of the kernel core functions at the specific interface between the driver programs and the kernel. To support this approach, we revisit and adapt the notion of common driver interface of [Edwards & Matassa 2002] by focusing the injection on the service/system calls made by the drivers to the kernel via the core functions. This way the definition of the fault injection experiments more thoroughly impact the interactions between the drivers and the kernel. Also, the errors provoked can simulate both the consequences of design faults or hardware-induced faults. It is worth noting that a subsequent and complementary work has been reported recently that adopt the same system model (explicit separation between the kernel and the drivers) and a similar fault model to study the impact of faulty drivers on a *Windows CE*-based system [Johansson & Suri 2005]. The main goal was to study the error propagation process to support the placement of protection wrappers (e.g., see [Fraser *et al.* 2003]), as was already carried out in [Arlat *et al.* 2002].

The material reported herein elaborates on the work reported in a previous paper [Albinet *et al.* 2004]. More detailed insights can also be found in [Albinet 2005]. The organization of the chapter is as follows. Section 2 describes the specific issues addressed by the approach we propose, in particular in the light of other work dealing with the characterization of operating system robustness. In Section 3, we briefly describe the various types of driver programs and introduce the specific interface considered to simulate faulty drivers for the *Linux* kernel: the *Driver Programming Interface*. Section 4 presents the experimental context: namely, the faultload, the workload and the measurements that define the conducted experiments and the testbed that supports these experiments: the RoCADE (Robustness Characterization Against Driver Errors) platform. Section 5 proposes a framework for supporting the analysis of the experimental results that can accommodate several end-user viewpoints. Section 6 presents a sample of results and illustrates how the framework proposed in Section 4 can support the analysis of the results. Finally, Section 7 concludes the chapter.

2. Context and Definition of the Approach

Due to their central role in the function of a computer system, operating systems are the privileged target for developing dependability benchmarks. Figure 1 depicts the software architecture of a computer system. In this chapter, due to the emphasis put on the analysis of the impact of the driver programs, the benchmark target (BT), according to the terminology put forward by the DBench project (e.g., see [Kalakech *et al.* 2004]), is the operating system kernel. Further drawing on that terminology, the whole figure describes the System Under Benchmark (SUB), i.e., the supporting environment and context within which the analyses are conducted.

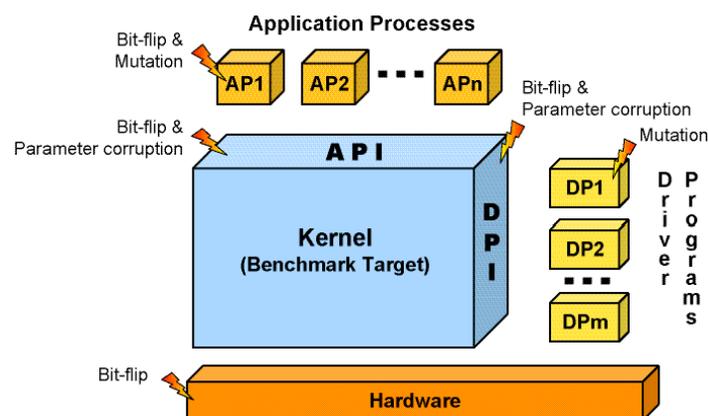


Figure 1 - Interactions between an operating system kernel and its environment

As shown on the figure, the kernel features three main interfaces with its environment. The first one is basically concerned with hardware interactions, while the other two are software related. The “lightning” symbols in Figure 1 identify possible locations where faults can be applied. The interfaces and related faults are briefly described as follows:

- 1) The bottom interface is primarily related to the hardware layer; the main interactions are made via the raising of hardware exceptions. Several studies (e.g., see [Arlat *et al.* 2002, Gu *et al.* 2003]) have been reported in which faults were injected by means of bit-flips into the memory of the SUB.
- 2) The interface at the top corresponds to the classical Application Programming Interface (API). The main interactions are made by means of system calls. A significant number of studies were reported that target the API to assess the robustness of the operating systems (e.g., under the form of code mutations [Durães & Madeira 2002]), by means of bit-flips [Jarboui *et al.* 2003] or by altering the system calls [Koopman & DeVale 1999, Kanoun *et al.* 2005b]).
- 3) The third type of interactions, that is the one we are specifically focusing on in this chapter, are made via the interface between the drivers and the kernel. Previous related work addressing the assessment of the kernel robustness has concentrated on drivers code mutation [Durães & Madeira 2003]. We propose an alternative approach where we explicitly consider the specific exchanges made between the drivers and the kernel, via what can be termed as the Driver Programming Interface (DPI). The precise definition of the DPI will be presented in Section 3.

Concerning the third item, we would like to refer to the work carried out in [Jarboui *et al.* 2003]. In this work, assertions issued from traces characterizing the actual erroneous behavior induced by faulty drivers were used to assess whether similar error patterns could be obtained by using several fault injection techniques (either bit flip or parameter corruption) at the API level. This study showed that API-level fault injection was not able to produce errors that were matching the error patterns provoked by real faults in drivers. This result further substantiates the need to conduct investigations specifically aimed at closely simulating the impact of faulty drivers. To this ends, we have concentrated our efforts on intercepting and corrupting the parameters of the system calls issued by the drivers at the DPI [Edwards & Matassa 2002]. Compared with the mutation of the code of the drivers used in [Durães & Madeira 2003], this approach allows for carrying out a more focused and efficient set of experiments that is suitable to thoroughly test the various kinds of interactions between the drivers and the kernel. The price to pay is a precise identification of the DPI upon which the faults are specified. However, it is worth noting that (as for the approaches targeting the API), such a preliminary analysis has to be carried out only once for each kernel family and can be reused for analyzing most drivers. More details on the types of faults considered are given in Section 4.1.

The interfaces depicted in Figure 1 (especially the API and the DPI) also provide suitable locations where to observe the consequences of the injected faults. At the API-level, the typical relevant behaviors include error codes returned to the calling application processes and kernel hangs. In practice, while a lot of exceptions are raised by the SUB at the hardware layer, for sake of efficiency they are often caught at the API when reported via the kernel. The DPI provides a privileged level of observation for the detailed characterization of the reactions of the kernel to the corrupted service calls issued by the drivers. The related measurements include error codes returned to the driver. Comprehensive measurements can also be made at the application-level, e.g., workload abort and completion. Then, time measurements can be collected to evaluate the workload execution in presence of faults [Kalakech *et al.* 2004].

In this chapter, we concentrate the conducted analyses on two levels of observation: the DPI and the API. The reported results only consider non-timed robustness measures, expressed as frequencies of occurrence of each considered outcomes. More details on the experimental measurements are provided in Section 4.

3. The Driver Programming Interface

In this section, we briefly recall the functional interactions that characterize the communication between the drivers and the kernel. This allows for the kernel functions and parameters involved into these interactions to be identified, and thus the DPI on which we define the types of faults to be injected.

3.1 Application Processes, Kernel and Drivers

The kernel and the drivers are executed in privileged mode whereas application processes execute in non-privileged mode in a restricted address space. This reduces the risk for an application process to corrupt the kernel addressing space. It is thus likely that the errors caused by a faulty application process mainly impact its own address space, and thus are limited to its execution. Nevertheless, this is different when an application process requires a service from the kernel by publishing a system call (e.g., interrupt #0x30 for *Pentium* or #0x87 for *PowerPC*).

Because the drivers execute in kernel space, any faulty behavior in a driver is thus much prone to impact the operation of the kernel. Due to the fact that it is not always possible to associate a “pedigree” to the whole set of drivers that can potentially be integrated, drivers are thus a potential threat for the kernel. This is further exacerbated by the programming languages (such as C language) that use pointer arithmetic without IMM (Integrated Memory Management). This applies to several popular general purpose operating systems (e.g., *Linux*, *Windows9x*, etc.). The drivers can also access the whole set of functions of the kernels, not only those that are used to carry out operations on the kernel space, but also on the application space.

3.2. The Various Drivers

An interesting comparative study of driver interfaces for several popular operating systems is presented in [Zaatar & Ouais 2002], as an initial step towards the standardization of the *Linux* driver interface. Irrespective of the different solutions adopted for a specific operating system family, in practice two main categories of drivers can be distinguished:

- Software drivers: they have no direct access to the hardware layer of the devices, but rather to an abstraction (e.g., tcp/ip stack, file system).
- Hardware drivers: they are concerned with hardware devices, either peripheral (network card, disk, printer, keyboard, mouse, screen etc.) or not (bus, RAM, etc.).

In both cases, the role of a driver is to provide an abstract interface for the operating system to interact with the hardware and the environment:

- More specifically, a driver is meant to implement a set of basic functions (read, write, etc.) that will activate peripheral devices.
- On top of drivers, the input-output instructions no longer depend on the hardware architecture.
- Drivers define when and how the peripheral devices interact with the kernel.

For example, in the case of a driver relying on polling, an application process issues a request, via a system call (`open`, `read` or `ioctl`), to access a peripheral device (network card, disk, printer, keyboard, mouse, screen etc.). The processor enters the supervisor mode — via a stub in the case of *Linux* — and executes the code of the driver corresponding to the proper operation. After completion of the operation, the driver frees the processor and the processor then resumes the execution of the application process in user mode.

Although device drivers may induce a strong influence on the kernel, as most of them are run in kernel mode, they are often developed by third parties and then integrated to the kernel after its distribution. This explains why it has been found that they significantly contributed to the failure of the operating system [Chou *et al.* 2001].

3.3. Specification of the DPI

The drivers make use of specific system calls (denoted symbols for dynamic module drivers in the case of *Linux*) in order to perform tasks. The most salient categories are depicted in Table 1.

Categories	Examples of Typical Symbols
Memory Management	<code>Kmalloc</code> , <code>kfree</code> , <code>free_pages</code> , <code>exit_mm</code> , ...
Interrupt Management	<code>add_timer</code> , <code>del_timer</code> , <code>request_irq</code> , <code>free_irq</code> , <code>irq_stat</code> , <code>add_wait_queue</code> , <code>_wait_queue</code> , <code>finish_wait</code> , ...
File System Management	<code>fput</code> , <code>fget</code> , <code>iput</code> , <code>follow_up</code> , <code>follow_down</code> , <code>filemap_fdatawrite</code> , <code>filemap_fdatawait</code> , <code>lock_page</code> , ...
Control Block Management	<code>blkdev_open</code> , <code>blkdev_get</code> , <code>blkdev_put</code> , <code>ioctl_by_bdev</code> , ...
Registration	<code>register_sysctl_table</code> , <code>unregister_sysctl_table</code> , <code>sysctl_string</code> , <code>sysctl_intvec</code> , ...
Others: Software interrupts, dma management, buffering management, resource handling, process management, interfaces, debug, miscellaneous “tools”	<code>raise_softirq</code> , <code>open_softirq</code> , <code>cpu_raise_softirq</code> , <code>dump_stack</code> , <code>ptrace_notify</code> , <code>current_kernel_time</code> , <code>sprintf</code> , <code>snprintf</code> , <code>sscanf</code> , <code>vsprintf</code> , <code>kdevname</code>

Table 1 - Outline of the categories of symbols for Linux

Each of these categories gathers a set of functions that are devoted to the programming of the kernel and drivers using execution privileges within the kernel address space. For example, in the case of *Linux*, functions allow for acquiring and releasing of an interrupt channel (`request_irq`, `free_irq`) and for retrieving the status of such a channel (`irq_stat`). These symbols form the basis for the development of drivers for managing the interrupt channels. All such symbols feature a calling protocol that is similar to the system calls of the *Linux* API. This is illustrated by the *signature* of the `request_irq` that is shown hereafter:

```
int request_irq(unsigned int irq,
               void (*handler)(),
               unsigned long irqflags,
               const char * devname,
               void *dev_id)
```

The `request_irq` function allocates a peripheral device to an interrupt channel. The function returns a success (error) code (an integer value) to inform the calling driver program of the proper (or not) handling of the reservation of the channel. The first argument `irq` is an unsigned integer that designates the channel to allocate. The second one `handler` is a pointer to the interrupt manager. The third one is an unsigned long integer that represents the flags that define the type of the reservation (exclusive, or not, etc.). `Devname` is the name of the peripheral device that is reserving the channel. The last parameter is a pointer to a “cookie” for the interrupt manager.

From more than thousand symbols (including functions, constants and variables), *Linux* release 2.4.18 includes about 700 kernel functions. Some are more used than others. The kernel functions devoted to memory reservation are definitely much more solicited than the ones attached to the handling of a *pcmcia* device. The types of the parameters being used in kernel programming are voluntarily restricted to integers (short or long, signed or unsigned) and pointers. We have referenced all these functions along with their signature, which allows for the number of parameters and their types to be specified for each symbol. These types are defined over a validity space (see extreme values in Section 4.1)

In the same way as the API that gathers all the available system calls issued by the application processes, the DPI gathers all the functions of the kernel that are available to be used by the drivers. These kernel symbols constitute the features offered to the developers in kernel mode.

4. The Experimental Framework

This section briefly describes the *execution profile* and the *measures* that are considered for the benchmarking analysis. The execution profile includes both, the *workload* (the processes that are executed to activate the drivers and the kernel) and the *faultload* (the set of faults that are applied during the fault injection experiments via the DPI). The experimental *measures*, that are meant to characterize the reaction and/or behavior of the kernel in presence of a faulty driver, are elaborated from a set of observations (*readouts* and *measurements*) that are collected during each experiment.

In order to illustrate how measurements can be used to derive useful measures, we will consider several dependability viewpoints according to the perception that different users can have from the observed behaviors. In the sequel of this section we concentrate first on the faultload, workload and measurements attributes and then we provide a brief description of the testbed (the RoCADE platform) set up to run the experiments. The way the measurements are exploited to derive relevant measures with an objective of dependability benchmarking will be addressed in Section 5.

4.1. The Faultload

For corrupting the parameters of the symbols of the DPI, we have used the SWIFI technique for its flexibility and ease of implementation. More precisely, in order to generate more efficient test conditions, we have focused the corruption of function parameters to a set of specific values. In particular, this procures a better control of the types of corruptions that are made, which significantly facilitates the interpretation of the results obtained. Faults are injected on each parameters of each relevant function of the DPI, as sketched by Figure 2.

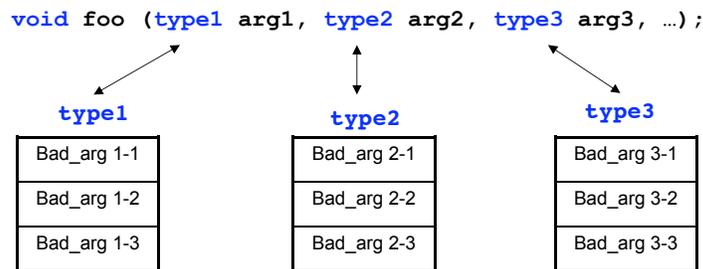


Figure 2 - Principle of corruption of the parameters of a function

The principle of the method is to intercept a function when it is called, to substitute the value of its parameters by a corrupted value and then to resume the execution of the function with this faulted value. The value that is substituted to the original value of the faulted parameter depends upon the type of the parameter. Table 2 shows the values considered for each relevant type. For the first three types, bounding and mid values are considered. For pointers the set of corrupted values are: NULL, a max bounding value and a random value.

Type	Bad_Arg 1	Bad_Arg 2	Bad_Arg 3
int	INT_MIN	0	INT_MAX (0x7FFFFFFF)
uint	0	INT_MIN (0x80000000)	ULONG_MAX (0xFFFFFFFF)
ushort	0	SHRT_MIN (0x8000)	USHRT_MAX (0xFFFF)
pointer	NULL	random()	All bits = 1 (0xFFFFFFFF)

Table 2 - The faulty parameters for each type

4.2. The Workload

In order to provoke the activation of the DPI by the driver programs, so as to mimic the nominal behavior, we rely on an indirect activation procedure by means of a workload applied at the level of the API.

We consider a synthetic and modular workload combining several activation processes, each targeting one (or several) of the drivers evaluated. Each application process carries out a set of elementary operations concerning a specific hardware or software driver component: i) de-installation of the target component that permits (only if the driver is currently used by the system) to start the test later on by registering the component, ii) (re-)installation of the component allowing for testing component registration, iii) series of requests meant for testing driver's operation, iv) de-installation where the unregistration of the component is tested, v) re-installation, whenever needed, in particular if the driver is mandatory for SUB's operation (e.g., network card or file system).

For example, in the case of a network card, the application process disables the network, unloads the network driver, reloads it, enables the network, runs a test on a private Ethernet network (Intranet), disables the network, unloads the driver, reloads it and enables the network.

The main differences between the application processes that form the workload concerns the specific requests to be applied to stimulate the driver.

In order to better assess the impact of the fault on the whole SUB, a subsequent workload execution is carried out after the fault has been withdrawn; this is particularly useful to improve the diagnosis in the cases when no outcome is observed as the result of the run when a fault is injected (the so-called "Silent" behavior as reported in the *CRASH* scale proposed in [Koopman & DeVale 1999]). In the reported experiments the workload that is executed for improving the diagnosis is the same as the workload used for the fault injection experiments. Accordingly, hereafter we will refer to it as the "replay" workload.

4.3. The Measurements

The goal of the set of experiments reported here is to determine the set of relevant observations to be incorporated into a prototype dependability benchmark, focusing on the robustness with respect to faulty drivers. Accordingly, to get relevant insights from the conducted experiments, it is necessary to obtain a good variety of results. To that respect, we have specified two levels of observation: i) *external or user-oriented*, that is meant to characterize the faulty behavior, as perceived at the level of the API, ii) *internal or peripheral device oriented*, that details the impact of the faults on the kernel, as perceived at the level of the DPI.

The *external* level includes the observation of the errors reported by the kernel to the application processes in the workload (exceptions, error codes, etc.). These observations can be augmented by a more *user-oriented* perception by means of observations directly related to the application processes (e.g., the *execution time* of the workload or the *restart time*). The *internal* level focuses on the exchanges between the kernel and the faulted driver. The specific observations made at each level as well as the related appraisals are depicted in Table 3.

The *error code* returned by a function of the kernel provides an essential insight on the impact of the fault on the intimate behavior of the kernel. Indeed, from a robustness viewpoint, the kernel symbol should be able to react to a service call including an argument with a corrupted value by returning an error code that matches the type of fault being injected. When a hardware *exception* is raised, while a process executes in the kernel address space, the kernel tries to abort the process or enters the “panic” mode. The consideration of Workload related events (WA or WI) allows for additional insights to be obtained, especially in cases when no error is notified by the kernel. In that respect, the “replay” workload that is executed after each run during which a parameter is corrupted, allows for the damage caused by the application of faulty call to be assessed by identifying whether the SUB was able to recover a stable state on its own or a specific restart is necessary.

Level	Event (= “1” when observed)	Appraisal good / bad
Internal	DPI Call Return Code(EC): Code returned by the kernel	1 / 0
External	Exception (XC): Processor’s exceptions observed at the API level	1 / 0
	Kernel Hang (KH): The kernel no longer replies to a request issued via the API	0 / 1
	Workload Abort (WA): The workload has been abruptly interrupted (some API service requests could not be made)	0 / 1
	Workload Incorrect (WI): The workload completes, but not all the return codes are “success”	0 / 1
	Workload Completion (WC): This event allows for the execution time of the workload programs to be measured	1 / 0
NB. Completion of the workload cannot be observed when the WA event is observed.		

Table 3 - Observation levels, events and appraisals

A *hang* of the kernel is diagnosed when the kernel is no longer replying to requests. Main reasons for such a blocking are either because it executes an infinite loop or it is waiting for an event while interrupts are masked. Such outcomes cannot be observed by the system and thus requires external monitoring.

While measuring the execution time of the workload programs provides useful information on the capacity of the kernel to handle the applications processes in presence of faults, and is thus a desirable feature from the benchmarking point of view. Due to the specific nature of the workload (synthetic workload), such a measurement was not carried out in the study described here. The interested reader can refer to the work reported in [Koopman & DeVale 1999]; the technique used therein can be applied to obtain the corresponding measurements.

4.4. The RoCADE Platform

Figure 3 describes the RoCADE (Robustness Characterization Against Driver Errors) platform that has been set up for conducting the experiments (only one target machine is shown). The experiments were carried out using a rack of four Intel Pentium machines each featuring 32 Mb of RAM and several commonly used peripheral devices: a hard disk, a floppy disk, a CD ROM, two network cards, a graphic card, a keyboard, etc. All four machines run the *GNU/Linux* distribution. Three of them are the Target Machines on which faults are injected and behaviors observed; each is supporting two versions of the *Linux* kernel: 2.2.20 and 2.4.18. The use of three target machines is meant to speed up the conduct of the experiments. The fourth machine (Control Machine) is connected to the target machines via a private Ethernet network to control the experiments and provide an external means for monitoring these machines. In particular, it is used to restart the target machines, should they be blocked after an experiment. Indeed, for sake of repeatability, for each experiment the SUB is restored to a specific (fault free) state.

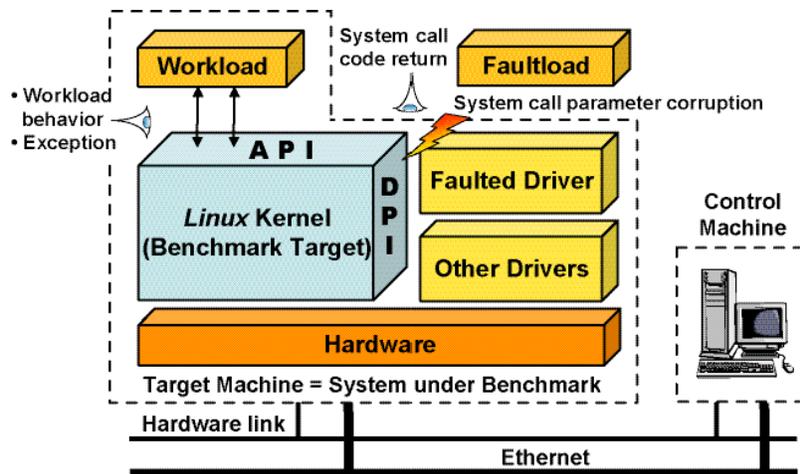


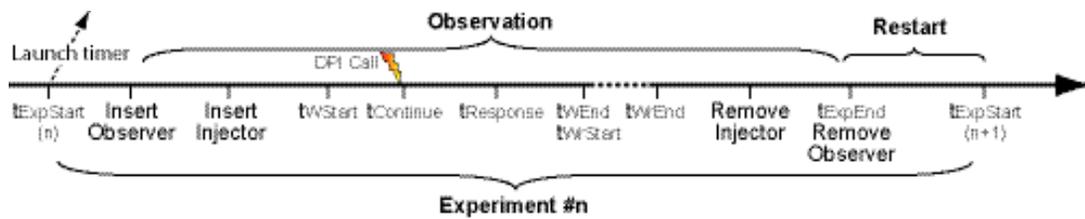
Figure 3 – Overview of the RoCADE platform

The injection of faulty parameters into each target machine is carried out via the RAM. The processor uses a stack residing in RAM to store various data, including the parameters of the calls to the functions of the DPI. This stack is accessible via the registers of the processor. At the same time, another area in the memory stores the instructions to be executed. When a DPI function is being used, the processor raises an interrupt. Upon occurrence of this interrupt the fault injection process takes over: it modifies a parameter in the stack and resumes the execution of the program. When the fault has been applied once, the fault injection process is disabled. The corruptions provoked in this way correspond to transient faults. This choice for the fault model illustrates the kind of pragmatic compromise one has to make among benchmarking properties (e.g., see [Kanoun *et al.* 2002]), namely here: fault representiveness and low intrusiveness.

In order to recognize the symbols used by the driver, we have developed scripts that automatically extract their names from the driver's object code file. Then, thanks to the list referencing all symbols, we can determine what faults can be injected on these symbols. Hence, all parameters of the selected functions are subjected to fault injection (according to all the fault types defined in Table 2). The codes returned after a system call are obtained with similar technique. The code returned by the symbol subjected to a fault is collected from the stack. In addition to these error codes, the symbols may also display other error messages, such as "blue screen" or "panic". Such error messages are collected at the end of each experiment.

At the start of each experiment that is indicated by the target machine, the control machine sets a timer. At the end of each experiment, the target machine is rebooted and it is expected to be able to retrigger this timer at the end of the reboot. If the timer overruns, the control machine provokes a hardware restart of the target machine. This situation is interpreted as a hang of the kernel. The hardware exceptions are collected via the log of the target machine. The duration of each fault injection experiment ranges from 2.5 to 5 minutes (the latter when a Kernel Hang occurs).

The diagram in Figure 4 presents the nominal scheduling of a fault injection experiment. The various important events are identified and described in the associated table, where related actions are also detailed.



IDs	Events	Actions
tExpStart Insertion X	System verification Set up of the modules of the tool and selection of the fault to be injected	Launch of <i>e2fsck</i> utility to <i>check the file system</i> integrity* Count down start (on the control machine) Insertion of a breakpoint
tWStart	Initiation of the workload	Start up of the workload
DPI call	Injection of the fault on the targeted kernel function call. Wait for (error) code returned by the kernel function	Raise of an interrupt and injection of the fault Insertion of the breakpoint for observing the returned code
tContinue	Resumption of the workload after execution of the function being faulted	Observation (internal) of the error code returned
tResponse	Observation of the events perceived externally	Collection of the results provided by the workload
TWEnd	Termination of the workload	Signaling of workload termination
tWrStart	Initiation of the replay workload	Start up of the replay workload
tWrEnd	Termination of the replay workload and observation of the related events perceived externally	Signaling of replay workload termination and collection of the related results provided
TExpEnd Removal X	End of current experiment	Removal of the modules of the tool and restart

* This proved a very useful procedure as in several instances the *file system* had been damaged due to the corruption of the system call.

Figure 4 - Scheduling of relevant events for an experiment

5. Interpretation of Measurements and Measures

The observations described in Section 4 offer a basis upon which various types of analyses can be carried out depending on how one assumes the impact of the combined behaviors observed from various dependability concerns. In practice, different interpretations of the measurements are possible depending on the specific context where the kernel is to be integrated. In particular, when one is favoring a safe behavior of the workload, then error notification via error code return or even kernel hangs might be proper or acceptable behaviors. Conversely, returned error codes or selective application process aborts are much more suited for cases when availability of the kernel is the desired property. This is further exacerbated in cases when several outcomes (e.g., error code return and hangs) are observed simultaneously within the same experiment run. So as to reliably account for various points of view, one has to carefully analyze such cases. It is worth pointing out that the types of analyses that we are proposing herein are in line and elaborate on the related study reported in [Rodríguez *et al.* 2002] and on the assessment framework used in [Durães & Madeira 2003].

5.1. Outcomes and Diagnoses

Table 4 provides an attempt at characterizing these issues. The first set of columns shows the possible outcomes (i.e., combinations of the events defined in Table 3): two categories are distinguished: error notification (explicit error reporting) and failure modes. The second part of the table illustrates how the outcomes when several events are collected per experiment can be diagnosed according to a set of simple criteria (order of occurrence of observed events and priority given either to error notification or failure modes).

#	Outcomes					First Event	Priority to	
	Notification		Failure Modes				Error Notification	Failure Modes
	EC	XC	WA	WI	KH			
O1	1	0	0	0	0	EC	EC	EC
O2	1	1	0	0	0	EC	EC+XC	EC+XC
O3	0	1	0	0	0	XC	XC	XC
O4	1	1	0	0	1	EC	EC+XC	KH
O5	1	0	0	0	1	EC	EC	KH
O6	0	1	0	0	1	XC	XC	KH
O7	0	0	0	0	1	KH	KH	KH
O8	1	1	1	X	1	EC	EC+XC	KH+WA
O9	1	0	1	X	1	EC	EC	KH+WA
O10	0	1	1	X	1	XC	XC	KH+WA
O11	0	0	1	X	1	KH	KH+WA	KH+WA
O12	0	0	0	0	0	No Obs.	No Obs.	No Obs.
O13	1	1	1	X	0	EC	EC+XC	WA
O14	1	0	1	X	0	EC	EC	WA
O15	0	1	1	X	0	XC	XC	WA
O16	0	0	1	X	0	WA	WA	WA
O17	1	1	0	1	0	EC	EC+XC	WI
O18	1	0	0	1	0	EC	EC	WI
O19	0	1	0	1	0	XC	XC	WI
O20	0	0	0	1	0	WI	WI	WI
O21	1	1	0	1	1	EC	EC+XC	WI+KH
O22	1	0	0	1	1	EC	EC	WI+KH
O23	0	1	0	1	1	XC	XC	WI+KH
O24	0	0	0	1	1	WI	WI+KH	WI+KH

Table 4 - Possible outcomes and diagnoses

First, it is worth noting that all events considered are not fully independent; accordingly, not all combinations are valid. In particular, this is the case for Workload Abort (WA) and Workload Incorrect (WI): indeed WA dominates WI, i.e., no WI can be observed when a WA has been diagnosed. This is identified by symbol “X” in Table 4. This explains why the table has only 24 rows. Among these, row O12 designates cases when none of the events has been observed. This is classical issue in testing and experimental studies when no impact is observed. This might be due to several alternatives (fault was not activated, error masked, etc.); we will come back on this in Section 6.1.

When several events are observed within the same experiment, various decisions can be made in order to categorize the outcomes. One usual approach is to give priority to the first event that has been observed. However, it is not always possible to have precise timing measurements for all events. Indeed, in some cases this

may require a sophisticated and heavy instrumentation (e.g., see [Rodríguez *et al.* 2003]), which might be out of the scope for a dependability benchmark that should be portable, minimally intrusive and cost effective.

Other alternatives include giving priority: i) to error notifications (e.g., error codes returned - EC and exceptions - XC) or ii) to the failure modes observed (Workload Abort, Workload Incorrect and Kernel Hang - KH). Considering the last two strategies, clearly the first one is optimistic (it assumes that notification will be able preempt and confine any subsequent impact) while the second one is rather pessimistic (the system is assumed to always fail, irrespective of the possible handling of the error ensuing the notification). In both cases, when multiple events are observed pertaining to the prioritized category, they are recorded for further analysis. The order of occurrence is also highlighted in the table. For example, when priority is given to failure modes, “WI + KH” in row O21 means that WI precedes KH. It is worth noting that, due to the way the considered events are collected, “Priority to Error Notification” closely matches “Priority to First Event”, because error notifications always precede all considered failure modes. Adopting a classification relying only on end-user perception would have resulted in discarding EC events. For example, in that case, O1 would have been merged into O12 and it would not be possible to discriminate O2 from O3.

5.2. Viewpoints and Interpretation

More elaborate interpretations can be defined that feature more dependability-oriented measures. We will consider three of such interpretations that correspond to three distinct contexts: Responsiveness of the Kernel (RK), i.e., maximize error notification, Availability of the Kernel (AK), i.e., minimize kernel hangs, and Safety of the Workload (SW), i.e., minimize the delivery of incorrect service by the applications processes.

The main rationale for the interpretation associated to **RK** is to positively consider outcomes gathering both notification events and failure modes. The fact that the kernel is able to notify an error is considered as positive, even when failure modes are observed at workload-level. Conversely, **AK** will rank differently the cases when either a KH or a WA is observed: indeed, the occurrence of a KH has a dramatic impact on the availability of the system, while an abort of the workload can be recovered more easily. The measure associated to **SW** characterizes the case when a safe behavior of the workload is required. Accordingly, we advocate that most favorable outcomes correspond to events prone to induce “fail-safe” or “fail-silent” behaviors, i.e., error notifications and kernel hangs, while workload abort is assumed to correspond to a critical event, and incorrect completion an even worse one. Nevertheless, as safety is typically an application-level property, alternative viewpoints could have been devised; in particular, from a “fail-fast” perspective, one may well consider that workload abort could be preferred to error notification.

Table 5 shows how these measures are linked to the outcomes described in Table 4. For sake of tractability, several outcomes are grouped into clusters that can be considered as equivalent with respect to a specific measure: each cluster characterizes a relevant “accomplishment level” for the considered measure. These clusters are ranked according to an increasing severity level (i.e., index 1 indicates most favorable case). We have appended labels (+) and (-) to explicitly indicate what we are considering as positive and negative clusters. However, we recommend keeping the data for each cluster so that a finer tuning of these categorizations is always possible. The rightmost column gives the rationale that defines the various clusters.

Viewpoint: Responsiveness/Feedback of the Kernel

	#	Outcomes [-O12]	Rationale
+	RK1	O1-O3	An error is notified by the kernel before the workload completes correctly
+	RK2	O4-O6, O8-O10, O13-O15, O17-O19, O21-O23	An error is notified by the kernel before a failure is observed
-	RK3	O16	No error is notified and the workload is aborted
-	RK4	O7, O11, O24	No error is notified and the kernel hangs
-	RK5	O20	No error is notified and the workload completes incorrectly

Viewpoint: Availability of the Kernel

	#	Outcomes [-O12]	Rationale
+	AK1	O1-O3	The workload completes correctly and an error is notified by the kernel
+	AK2	O13-O20	The workload is aborted or completes incorrectly
-	AK3	O4-O7	The workload completes correctly and the kernel hangs
-	AK4	O8-O11, O21-O24	The workload is aborted or completes incorrectly and the kernel hangs

Viewpoint: Safety of the Workload

	#	Outcomes [-O12]	Rationale
+	SW1	O1-O3	The workload completes correctly and an error is notified by the kernel
+	SW2	O4-O7	The workload completes correctly and the kernel hangs
+	SW3	O8-O11, O13-O16	The workload is aborted or the kernel hangs
-	SW4	O21-O24	The workload completes incorrectly and the kernel hangs
-	SW5	O17-O20	The workload completes incorrectly and the kernel does not hang

Table 5 - Viewpoints and dependability measures

6. Results and Analyses

This section illustrates how the insights one can get from the measurements obtained vary according to the priorities or dependability measures that are considered. We present and analyze a restricted set of results obtained with the RoCADE platform for three representative drivers running on the *Linux* kernel. We restrict the presentation of the results to a selected set, in order to facilitate the exposition of the analyses. We voluntarily emphasize two drivers supporting the network card (namely the SMC-ultra and the Ne2000). Network drivers account for the largest part of the code among the drivers and whose size is increasing the most [Godfrey & Tu 2000]; we consider also another driver (namely, SoundBlaster) that ranges in the mid-size category. An additional set of drivers has been tested (e.g., file system, process memory, etc.).

The main goal that supports the selection of this set of results is to be able to carry out the following types of analyses on:

- Two distinct drivers running on the same version of the kernel: SB 2.2 and SMC 2.2;
- Two implementations of the same functionality running on the same kernel: SMC 2.4 and NE 2.4;
- The same driver¹ running on two different versions of the kernel: SMC 2.2 and SMC 2.4.

Based on the workload and fault types considered, about 100 experiments were carried out for each driver/kernel combination.

¹ It is worth pointing out that the code of the driver is adapted to fit each version of the kernel.

6.1. Basic Results and Interpretation

Table 6 illustrates the distribution of the basic results obtained when considering the “first event” approach to diagnose outcomes for which multiple events were collected.

Driver	Not Act.	No Obs.	EC	XC	KH	WA	WI
SB 2.2	0%	18%	47%	22%	3%	1%	9%
SMC 2.2	7%	22%	19%	23%	21%	0%	9%
SMC 2.4	17%	17%	21%	34%	11%	0%	0%
NE 2.4	14%	10%	15%	30%	17%	0%	13%

Table 6 - Distribution of events according to first event collected

In addition to the specific events previously defined (see Table 4), two interesting outcomes are included:

- Not Activated (Not Act.): injected faults could not be activated (i.e., the workload was not able to activate the function on which the fault was meant to be injected).
- No Observation (No Obs.): none of the notification or failure modes events were observed; of course when the fault is not activated, none of these events can be observed.

The proportion of “Not Activated” cases varies significantly, both among the tested drivers and *Linux* versions — from 0% (SB 2.2) to 17% (SMC 2.4). The fact that in most cases non-null ratios are observed, means that the respective workloads have to be improved from a *testability* viewpoint — more precisely *controllability* here. However, these rates are much lower than those reported in related studies on the *Linux* kernel (e.g., see [Gu *et al.* 2003]). To our understanding, this better *controllability* is most likely due to the fact that, in our case, faults are targeting the parameters of the system calls made by the driver, rather than the flow of execution of the whole kernel. In the sequel, for further analyses, we will normalize the results presented with respect to experiments where faults were actually activated.

As already pointed out, the interpretation of the “no observation” cases is highly subject to the specific context where the analysis is conducted. These outcomes may be counted either as positive or negative depending on the responsiveness/safety/availability viewpoints. However, as is commonly accepted in testing scenarios, uncertainties still remain about the real situations that such an outcome describe. Accordingly, we have preferred to adopt a conservative approach that consists in ignoring these outcomes. Besides, the “replay” mode has been devised to increase the confidence in our analyses, a “No Obs” outcome probably still reveals a lack of *observability* of the tests conducted. But, such an outcome may also be due to controllability-related problems: the kernel does not (or no longer) use(s) the faulted parameter, the faulted parameter has no impact on the kernel, or the error provoked is masked (in our case, injecting a “0” value on a parameter already equal to “0”, etc.). However, although the “No Obs.” ratios reported are higher than the “Not activated”, the values are significantly lower than the ones presented in [Gu *et al.* 2003].

Figure 5 illustrates the relative distribution among the events observed still considering the “first event” collected — which is a classical approach in most related experimental studies. A quick examination of these results shows a very low proportion of Workload Aborts for all tests conducted. The results also reveal that a large percentage of experiments are notified by the kernel (this includes the Error Code and Exception events). Should it be possible to handle equally both types notifications, then as the provision of an error code usually features a lower latency such a notification would be preferable to an exception in order to carry out a successful

recovery action. Accordingly, in that respect, the results for SBL 2.2 are more positive than those observed for the experiments concerning network card drivers. However, adopting a end-user perspective would lead to a different assessment: indeed, in that case, only exceptions would actually matter.

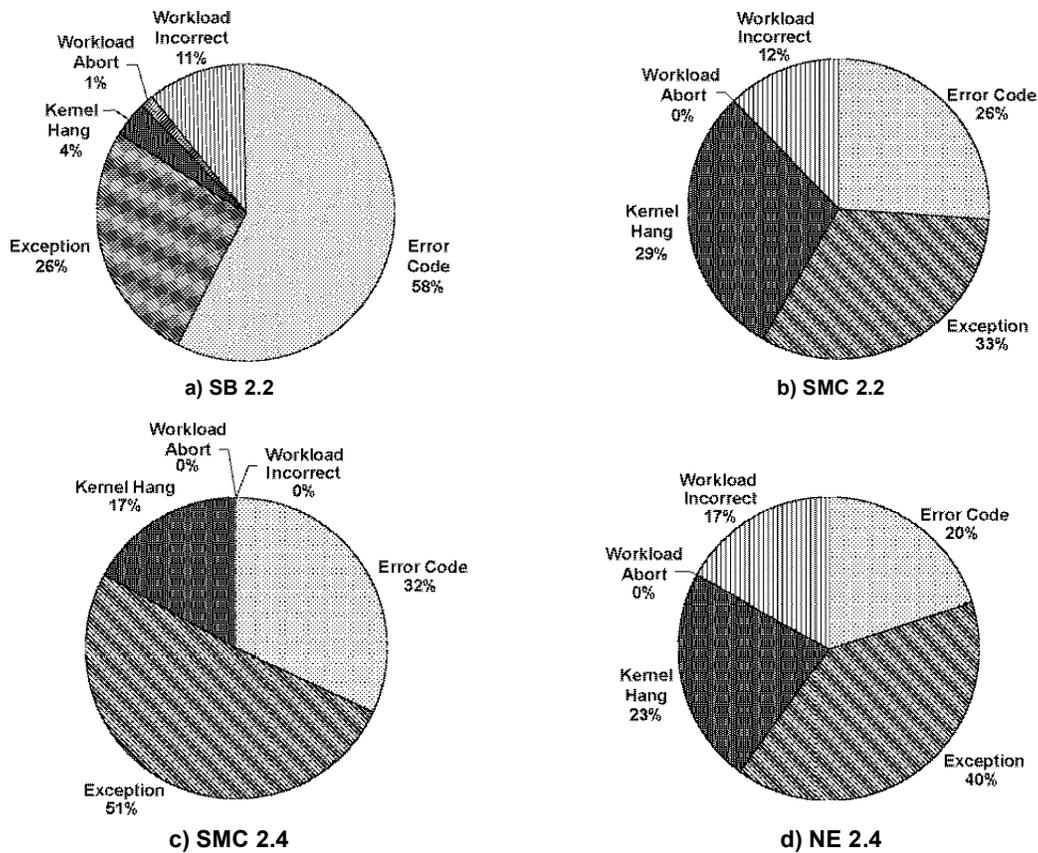


Figure 5 - Distribution among observed events according to first event collected

The comparison of the results obtained for the SMC driver for the two releases indicates clearly an improvement of the robustness for SMC 2.4 due to the increased percentage of exceptions raised. This results in a reduction of the ratios of Kernel Hangs and more importantly, in the “disappearance” of critical cases where a Workload Incorrect event was reported. Indeed, due to the precedence in the collection of the events, the fact that a WI event is counted as a first event means that neither a notification has been made nor an abort has been observed. It is also very likely to be the only event to be collected, unless a hang has occurred after the end of the workload (such cases are actually very seldom). But, in practice, a deeper analysis of the data collected is necessary to ascertain this statement.

6.2. Impact of the Comprehensive Viewpoints

In this section we revisit the observations made during the conducted experiments in the light of the three comprehensive viewpoints defined in Section 5. Figure 6 summarizes the corresponding measures for the four series of experiments reported here. In each case, the percentages of the various clusters that support the corresponding measure are detailed. It is important to note that the clusters corresponding to the most positive outcomes appear on the top of the histograms (light grey) while the critical ones are at the bottom (darker shade). Here we consider the set of outcomes defined in Table 4.

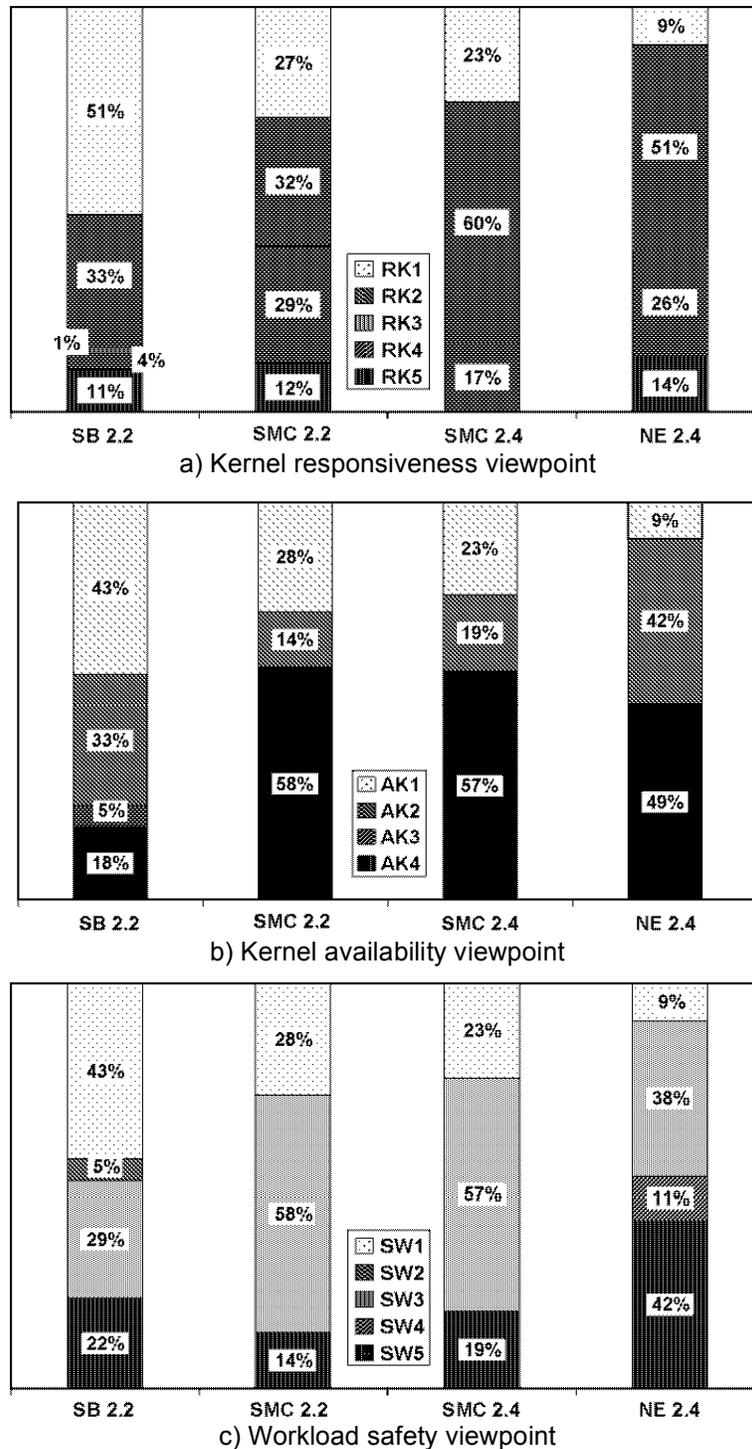


Figure 6 - Interpretation of the results according to the considered viewpoints

Let us consider first the kernel responsiveness (RK) viewpoint. Here we assume that RK1 and RK2 form the most positive outcomes (Table 5). The distribution observed for SB 2.2 indicates a very positive behavior: 84 % of the outcomes observed correspond to error notifications (RK1=51% + RK2=33%). However, among the 16% of outcomes for which a failure mode was observed without prior notification, more than 2/3 corresponds to WI events (RK5=11%). The remaining 1/3 is dominated by KH events and few WA events. It is worth noting that

this is the only set of experiments for which not notified WA events have been diagnosed. The network driver considered for this version of *Linux* (SMC 2.2) features a much less positive behavior. For example, the distribution shows that 41% of the outcomes observed correspond to failure modes without prior notification. This is mainly due to KH events (RK4=29%), the RK5 cluster (not notified WI events) amounts to a similar 12%. This also means that almost 30% of the workload failures that are not notified led to an incorrect completion. The results shown for SMC 2.4 indicate that the evolution to release 2.4 has significantly improved the behavior: the percentage of failures without prior notification is reduced to 17% and corresponds to KH events only. Globally, more than 3/4 (60/77) of the failure modes observed are preceded by an error notification. This confirms the observations already made on the basis of the analysis of the pie charts displaying the distributions of the first event collected (Figure 5). The results for NE 2.4 indicate a much lower error notification ratio, which is similar to the one reported for the SMC 2.2 case.

For the kernel availability viewpoint (AK), the most critical issue is characterized by a KH event, because this has a dramatic impact on the ability to keep delivering the service. This is why AK3 and AK4 are considered as the most critical clusters. The results shown for SB 2.2 indicate that faults have also a significant impact with respect to availability. Indeed, cases when a KH event is observed amount to 23% and in more than 78% of these cases a WI event is also observed. The results also show that faults associated to network drivers have consistently a very significant impact: about 1/2 of the fault injection experiments conclude with a KH event.

Concerning the workload safety viewpoint (SW), what matters most is the ability to avoid the delivery of incorrect results. This is why SW4 and SW5 are considered as the most critical clusters. The results shown for SB 2.2 suggest a much less positive behavior than what was deduced from the analysis of the results from the RK viewpoint: the occurrence of the most severe cluster SW5 (WI and no KH) amounts to 22%. In addition, it is possible to mention that the significant improvement observed with respect to responsiveness (error notification) between SMC 2.2 and SMC 2.4 has not impact (actually slightly the opposite) in reducing the WI events. Such a behavior can be explained by the fact that most additional error notifications correspond to exceptions, rather than error codes returns (see Figure 5). As a matter of fact, such exceptions signal already severe erroneous behavior. The rather poor behavior observed with respect to responsiveness is also confirmed by the safety viewpoint: in 53% of the cases (SW5=42% + SW4=11%) the observed outcome is a WI event.

Figure 7 illustrates how these various viewpoints — and the associated properties — can be used by system integrators in making a decision whether to incorporate a driver into their system. The histograms plot the percentage for cases when these properties were not verified (i.e., the cases corresponding to the clusters labeled with index “-” in Table 5) when considering the experiments involving the tested network drivers. In this case, the figures being considered explicitly account for the ratios of “No Obs.” that already appeared in Table 6. It is worth noting that these ratios contribute negatively to the evaluation of the responsiveness property (i.e., RK is not verified²); however, they correspond to the verification of AK and SW.

² This is consistent with the rationale underlying this viewpoint (i.e., a reaction from the kernel is expected in presence of activated faults). Still, considering the complete “No Obs.” ratio as contributing to a deficiency of the RK property might lead to a pessimistic assessment. This is why the related percentages are explicitly recalled in the figure.

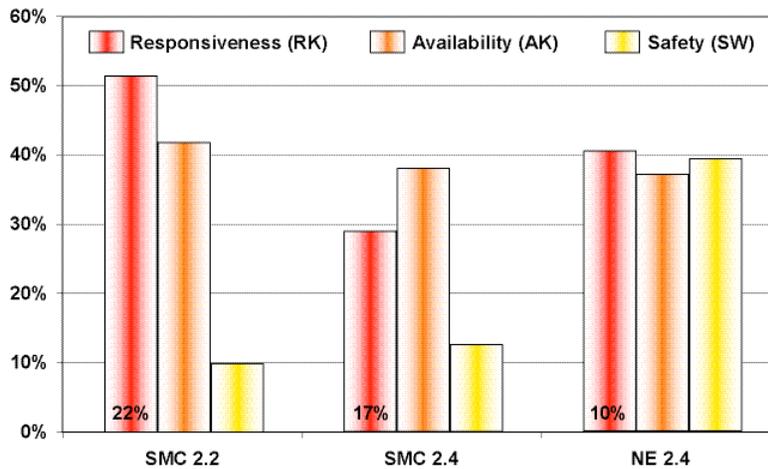


Figure 7 - Comparison of the property deficiencies induced by the network card drivers

The histograms concerning the two versions of the SMC driver clearly illustrate that the significant decrease in lack of error signaling obtained for version 2.4 does not result in a significant reduction in the weaknesses with respect to the other viewpoints (actually, a slight increase is observed for safety): the improvement in the coverage procured by the error detection mechanisms was not accompanied by an improvement in the handling of error signals. For example, the application of the concept of *shadow driver* reported in [Swift *et al.* 2004] would help improve the behavior by complementing such a *problem revealing only* strategy with a specific low-level *recovery* strategy. During normal operation, the shadow tracks the state of the real driver by monitoring all communication between the kernel and the driver. When a failure occurs, the shadow driver substitutes *temporarily* the real driver, servicing requests on its behalf, thus shielding the kernel and applications from the failure. The shadow driver then restores the failed driver to a state where it can resume processing requests. It is also interesting to observe that, while the network driver Ne 2000 features similar or slightly better behaviors than the SMC driver with respect to responsiveness and availability, it exhibits a much poorer behavior with respect to safety. This reflects the fact that very distinct implementation choices were made for these two drivers.

6.3. Detailed Analysis

In addition to the comparison of distinct benchmark targets (kernels and related device driver interfaces) with respect to various assessment properties as was reported in the preceding sub-section, it is also possible to conduct more in-deep analysis of a given target in presence of faults. As was shown in [Arlat *et al.* 2002], such detailed analysis is instrumental in order to devise and incorporate suitable protection mechanisms.

As an example, Figure 8 plots the frequency of the experimental results concerning the external behavior of the workload and of the kernel (including also the “No Observation” category), so that the total for the four categories sum up to the number of injections carried out for each function. Due to the fault model used (see Table 2), this number is directly related to the number and the types of the parameters of the function.

These components are often crafted by third-party developers and then integrated within the operating system. This whole process is not always well mastered, as evidenced by the vast consensus that attributes a large proportion of operating system failures to driver malfunctions.

The work reported in this chapter proposed a practical approach to characterize the robustness of operating systems with respect to faulty device drivers. In order to facilitate the conduct of fault injection experiments we have introduced the notion of Driver Programming Interface (DPI) that precisely identifies the interface between the drivers and the kernel, under the form of a set of kernel functions. In the same way as the API is used to simulate the consequences of faulty application processes, the DPI provides a suitable interface for simulating the potential erroneous behaviors induced by a faulty driver. In practice, we have used a SWIFI technique to corrupt the parameters of these functions. In order to collect relevant outcomes for a detailed characterization of the faulty behaviors, we have considered both internal (error codes returned by the kernel) and external measurements (e.g., exceptions raised, kernel hangs, and workload behavior).

To analyze the experimental results, we have proposed a comprehensive framework for interpreting the results that accounts for several dependability viewpoints. We have considered three viewpoints, namely, responsiveness of the kernel (maximize error notification), availability (minimize kernel hangs) and safety of the workload (minimize delivery of incorrect service). They provide a practical means for analyzing three different facets of the dependability requirements that one can be expecting from a robust operating system, either simultaneously or individually.

In order to illustrate and assess our approach, we have set up an experimental platform RoCADE (Robustness Characterization Against Driver Errors). We have focused here on the series of experiments conducted on two releases of the *Linux* kernel and on three drivers (sound: sound blaster and network: SMC and Ne2000). The analyses carried out have evidenced that although the sound blaster driver got a very good rating according to responsiveness, it exhibited a poor behavior with respect to the safety and availability viewpoints. The experiments conducted on the SMC driver were able to reveal a significant improvement with respect to responsiveness between the two releases considered, but this did not result in any improvement from safety and availability viewpoints. Finally, we identified a slightly better behavior concerning availability for the experiments conducted on the Ne 2000 driver than for those on the SMC driver, while the opposite was obtained from safety and responsiveness.

The results we have obtained and the analyses we have carried out thanks to RoCADE comfort us in the interest and viability of the proposed methodology. The whole approach can thus be considered as a sound basis on which to develop a set of practical dependability benchmarks focusing on the characterization of the impact of faulty drivers on the behavior of an operating system kernel. As was witnessed by the insights gained from the measures obtained, while the proposed frame is primarily geared towards the characterization of kernel behaviors, it is also suitable to support the choice of drivers to be associated to a given kernel.

We consider the fact that a large proportion of error codes had been observed (especially as first collected event) as a positive result in order to perform a detailed characterization of the erroneous behaviors induced by the corrupted parameters. In addition, these codes form a useful basis on which specific error handling could be implemented. Another recommended approach to restrict the impact of faulty drivers would be to enforce a clear separation between the driver address space and the kernel address space (e.g., see [Härting *et al.* 1997]). The use of specific languages excluding pointer arithmetic and explicitly including IMM (e.g., see [Réveillère &

Muller 2001]) is another promising approach to develop more robust drivers. More recently, several proposals have been made to attain a clear separation of concern using virtual machine constructs, e.g., see [Fraser *et al.* 2004, LeVasseur *et al.* 2004]. The contemporary Nooks approach and its extension, under the form of *shadow drivers* [Swift *et al.* 2004] offer other attractive approaches.

Finally, it is worth pointing out that the notion of DPI (Driver Programming Interface) that we have advocated and defined in order to structure the conducted experiments, matches very well the concept of separation of concerns that is underlying several frameworks that were proposed recently — both by academic studies (e.g., see [Swift *et al.* 2004]) and by an increasingly number of operating system and hardware manufacturers. Let us simply mention the various CDI (Common Driver Interface), DDI (Device Driver Interface) or DKI (Driver Kernel Interface) proposals that are been put forward for several operating systems. Among these initiatives, the Extensible Firmware Interface (EFI) that was recently promoted by the Unified EFI Forum³ as an emerging standard, deserves special attention. The EFI defines a new model for the interface between operating systems and platform firmware. The UEFI is primarily meant to provide a standard environment for booting an operating system. Nevertheless, the data tables (containing platform-related information, plus boot and runtime service calls) that implements it can be useful also to facilitate runtime access to internal variables and thus better structure the design of device drivers. Accordingly, it should be possible to reuse the principles underlying the DPI identified herein and/or to adapt them easily in the forthcoming arena that this emerging standard is promising for structuring the interactions between the operating systems and the related hardware layers, including the device drivers.

Acknowledgement

This work was partly supported by the European Commission (Project IST-2000-25425: DBench and Network of Excellence IST-026764: ReSIST). Arnaud Albinet was also supported in part by the *Réseau d'Ingénierie de la Sûreté de fonctionnement* (Network of Dependability Engineering); he has now joined Siemens VDO, Toulouse, France.

References

- [Albinet 2005] A. Albinet, *Dependability Characterization of Operating Systems in presence of Faulty Drivers*, PhD Dissertation, National Polytechnic Institute, Toulouse, 2005 (In French - also LAAS Report 05-248).
- [Albinet *et al.* 2004] A. Albinet, J. Arlat and J.-C. Fabre, “Characterization of the Impact of Faulty Drivers on the Robustness of the *Linux* Kernel”, in *Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN-2004)*, (Florence, Italy), pp.867-876, IEEE CS Press, 2004.
- [Arlat *et al.* 2002] J. Arlat, J.-C. Fabre, M. Rodríguez and F. Salles, “Dependability of COTS Microkernel-Based Systems”, *IEEE Transactions on Computers*, 51 (2), pp.138-163, February 2002.
- [Avižienis *et al.* 2004] A. Avižienis, J.-C. Laprie, B. Randell and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing”, *IEEE Transactions on Dependable and Secure Computing*, 1 (1), pp.11-33, Jan.-March 2004.

³ <http://www.uefi.org>

- [Brown & Patterson 2000] A. Brown and D. A. Patterson, “Towards Availability Benchmarks: A Case Study of Software RAID Systems”, in *Proc. 2000 USENIX Annual Technical Conference*, (San Diego, CA, USA), USENIX Association, 2000.
- [Carreira *et al.* 1998] J. Carreira, H. Madeira and J. G. Silva, “Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers”, *IEEE Transactions on Software Engineering*, 24 (2), pp.125-136, February 1998.
- [Chou *et al.* 2001] A. Chou, J.-F. Yang, B. Chelf, S. Hallem and D. Engler, “An Empirical Study of Operating System Errors”, in *18th Symposium on Operating Systems Principles*, (Chateau Lake Louise, Banff, Canada), ACM Press, 2001, <http://www.cs.ucsd.edu/sosp01>.
- [Durães & Madeira 2002] J. Durães and H. Madeira, “Emulation of Software Faults by Selective Mutations at Machine-code Level”, in *Proc. 13th Int. Symp. on Software Reliability Engineering (ISSRE-2002)*, (Annapolis, MD, USA), pp.329-340, IEEE CS Press, 2002.
- [Durães & Madeira 2003] J. Durães and H. Madeira, “Mutidimensional Characterization of the Impact of Faulty Drivers on the Operating Systems Behavior”, *IEICE Transactions on Information and Systems*, E86-D (12), pp.2563-2570, December 2003.
- [Edwards & Matassa 2002] D. Edwards and L. Matassa, “An Approach to Injecting Faults into Hardened Software”, in *Proc. Ottawa Linux Symposium*, (Ottawa, ON, Canada), pp.146-175, 2002.
- [Fraser *et al.* 2004] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield and M. Williamson, “Safe Hardware Access with the Xen Virtual Machine Monitor”, in *First Workshop on Operating System and Architectural Support for the On-Demand IT Infrastructure (OASIS)*, (Boston, MA, USA), 2004.
- [Fraser *et al.* 2003] T. Fraser, L. Badger and M. Feldman, “Hardening COTS Software with Generic Software Wrappers”, in *Foundations of Intrusion Tolerant Systems – Organically Assured and Survivable Information Systems (OASIS)* (J. H. Lala, Ed.), pp.399-413, IEEE CS Press, 2003.
- [Godfrey & Tu 2000] M. W. Godfrey and Q. Tu, “Evolution in Open Source Software: A Case Study”, in *Proc. IEEE Int. Conf. on Software Maintenance (ICSM-200)*, (San Jose, CA, USA), pp.131-142, IEEE CS Press, 2000.
- [Gu *et al.* 2003] W. Gu, Z. Kalbarczyk, R. K. Iyer and Z. Yang, “Characterization of Linux Kernel Behavior under Errors”, in *Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN-2003)*, (San Francisco, CA, USA), pp.459-468, IEEE CS Press, 2003.
- [Härting *et al.* 1997] H. Härting, M. Ohmuth, J. Liedtke, S. Schönberg and J. Wolter, “The Performance of μ -Kernel-Based Systems”, in *Proc. 16th ACM Symp. on Operating Systems Principles (SOSP-16)* (Saint-Malo, France), pp.66-77, 1997.
- [Jarboui *et al.* 2003] T. Jarboui, J. Arlat, Y. Crouzet, K. Kanoun and T. Marteau, “Impact of Internal and External Software Faults on the Linux Kernel”, *IEICE Transactions on Information and Systems*, E86-D (12), pp.2571-2578, December 2003.
- [Johansson & Suri 2005] A. Johansson and N. Suri, “Error Propagation Profiling of Operating Systems”, in *Proc. IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN-2005)*, (Yokohama, Japan), pp.86-95, IEEE CS Press, 2005.
- [Kalakech *et al.* 2004] A. Kalakech, T. Jarboui, J. Arlat, Y. Crouzet and K. Kanoun, “Benchmarking Operating System Dependability: Windows 2000 as a Case Study”, in *Proc. 10th Pacific Rim Int. Symp. on Dependable Computing (PRDC-2004)*, (Papeete, French Polynesia), pp.261-270, IEEE CS Press, 2004.
- [Kanoun *et al.* 2002] K. Kanoun, H. Madeira and J. Arlat, “A Framework for Dependability Benchmarking”, in *Supplemental Volume of the 2002 Int. Conf. on Dependable Systems and Networks (DSN-2002) - Workshop on Dependability Benchmarking*, (Washington, DC, USA), pp.F.7-F.8, 2002, see also <http://www.laas.fr/DBench>.

- [Kanoun *et al.* 2005a] K. Kanoun, H. Madeira, M. Dal Cin, F. Moreira and J. C. Ruiz Garcia, “DBench (Dependability Benchmarking)”, in *5th European Dependable Computing Conference (EDCC-5) - Project Track*, (Budapest, Hungary), 2005, Available as LAAS Report n°05197, 4p., see <http://www.laas.fr/DBench>.
- [Kanoun *et al.* 2005b] K. Kanoun, Y. Crouzet, A. Kalakech, A. E. Rugina and P. Rumeau, “Benchmarking the Dependability of Windows and Linux using Postmark Workloads”, in *Proc. 16th IEEE Int. Symp. on Software Reliability Engineering (ISSRE 2005)*, (Chicago, IL, USA), pp.11-20, IEEE CS Press, 2005.
- [Koopman & DeVale 1999] P. Koopman and J. DeVale, “Comparing the Robustness of POSIX Operating Systems”, in *Proc. 29th Int. Symp. on Fault-Tolerant Computing (FTCS-29)*, (Madison, WI, USA), pp.30-37, IEEE CS Press, 1999.
- [LeVasseur *et al.* 2004] J. LeVasseur, V. Uhlig, J. Stoess and S. Götz, “Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines”, in *Proc. 6th Symp. on Operating Systems Design and Implementation (OSDI '04)*, (San Francisco, CA, USA), pp.17-30, USENIX Association, 2004.
- [Madeira *et al.* 2002] H. Madeira, R. Some, F. Moreira, D. Costa and D. Rennels, “Experimental Evaluation of a COTS System for Space Applications”, in *Proc. Int. Conference on Dependable Systems and Networks (DSN-2002)*, (Washington, DC, USA), pp.325-330, IEEE CS Press, 2002.
- [Marsden *et al.* 2002] E. Marsden, J.-C. Fabre and J. Arlat, “Dependability of CORBA Systems: Service Characterization by Fault Injection”, in *Proc. 21st Int. Symposium on Reliable Distributed Systems (SRDS-2002)*, (Osaka, Japan), pp.276-285, IEEE CS Press, 2002.
- [Mukherjee & Siewiorek 1997] A. Mukherjee and D. P. Siewiorek, “Measuring Software Dependability by Robustness Benchmarking”, *IEEE Transactions on Software Engineering*, 23 (6), pp.366-378, June 1997.
- [Murphy & Levidow 2000] B. Murphy and B. Levidow, “Windows 2000 Dependability”, in *Digest Workshops and Abstracts of the 2000 Int. Conference on Systems and Networks (DSN-2000)*, (New York, NY, USA), pp.D.20-D.28, 2000.
- [Réveillère & Muller 2001] L. Réveillère and G. Muller, “Improving Driver Robustness: An Evaluation of the Devil Approach”, in *Proc. Int. Conference on Dependable Systems and Networks (DSN-2001)*, (Göteborg, Sweden), pp.131-140, IEEE CS Press, 2001.
- [Rodríguez *et al.* 2002] M. Rodríguez, A. Albinet and J. Arlat, “MAFALDA-RT: A Tool for Dependability Assessment of Real Time Systems”, in *Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN-2002)*, (Washington, DC, USA), pp.267-272, IEEE CS Press, 2002.
- [Rodríguez *et al.* 2003] M. Rodríguez, J.-C. Fabre and J. Arlat, “Building SWIFI Tools from Temporal Logic Specifications”, in *Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN-2003)*, (San Francisco, CA, USA), pp.95-104, IEEE CS Press, 2003.
- [Swift *et al.* 2004] M. M. Swift, M. Annamalai, B. N. Bershad and H. M. Levy, “Recovering Device Drivers”, in *Proc. 6th ACM/USENIX Symposium on Operating Systems Design and Implementation*, (San Francisco, CA, USA), 2004, <http://nooks.cs.washington.edu>.
- [Tsai *et al.* 1996] T. K. Tsai, R. K. Iyer and D. Jewitt, “An Approach Towards Benchmarking of Fault-Tolerant Commercial Systems”, in *Proc. 26th Int. Symp. on Fault-Tolerant Computing (FTCS-26)*, (Sendai, Japan), pp.314-323, IEEE CS Press, 1996.
- [Vieira & Madeira 2003] M. Vieira and H. Madeira, “Benchmarking the Dependability of Different OLTP Systems”, in *Proc. IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN-2003)*, (San Francisco, CA, USA), pp.305-310, IEEE CS Press, 2003.
- [Zaatar & Ouais 2002] W. Zaatar and I. Ouais, “A Comparative Study of Device Driver APIs: Towards a Uniform Linux Approach”, in *Proc. Ottawa Linux Symposium*, (Ottawa, ON, Canada), pp.407-413, 2002.
- [Zhu *et al.* 2003] J. Zhu, J. Mauro and I. Pramanick, “Robustness Benchmarking for Hardware Maintenance Events”, in *Proc. IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN-2003)*, (San Francisco, CA, USA), pp.115-122, IEEE CS Press, 2003.

Dependability Evaluation of Web Service-Based Processes*

László Gönczy¹, Silvano Chiaradonna², Felicita Di Giandomenico²,
András Pataricza¹, Andrea Bondavalli³, and Tamás Bartha¹

¹ DMIS, Budapest University of Technology and Economics
Magyar Tudósok krt. 2. H-1117, Budapest, Hungary
gonczy@mit.bme.hu

² ISTI-CNR, Via G. Moruzzi 1, I-56124 Pisa, Italy
+39 50 315 2904

{silvano.chiaradonna, felicita.digiandomenico}@isti.cnr.it

³ DSI - Università di Firenze, Viale Morgagni 65, I-50134 Firenze, Italy
+39 55 479 6776
bondavalli@unifi.it

Abstract. As Web service-based system integration recently became the mainstream approach to create composite services, the dependability of such systems becomes more and more crucial. Therefore, extensions of the common service composition techniques are urgently needed in order to cover dependability aspects and a core concept for the dependability estimation of the target composite service. Since Web services-based workflows fit into the class of systems composed of multiple phases, this paper attempts to apply methodologies and tools for dependability analysis of Multiple Phased Systems (MPS) to this emerging category of dependability critical systems. The paper shows how this dependability analysis constitutes a very useful support to the service provider in choosing the most appropriate service alternatives to build up its own composite service.

1 Introduction

Recently, the main paradigm of creating large scale information systems is shifting more and more towards integrating services instead of integrating components as in traditional technologies. Open standards like Web Service Description Language (WSDL) assure system interoperability. This integration and development paradigm is called Service Oriented Architecture (SOA). The top level description of a SOA process describes the main business logic and it is usually very close to the traditional business process models (BPM). Recent development tools provide a quite powerful support for functional service integration but they lack the support of the description and analysis of the non-functional aspects in the system. However, service level

* This work was partially supported by the HIDDENETS European project (IST-FP6-STREP-26979) and the "Quality of Service and Dependable Computer Networks" Project of the Italian-Hungarian Intergovernmental S&T Cooperation Programme.

integration raises new problems as the service provider is composing its main services from elementary services as building blocks without having a complete control over them. Thus, the result of the main service may be invalidated by simple faults and errors in imported services. Dependability analysis has to focus on creating a system-wide dependability model of the component models and evaluating the impact of the faults in the individual components, including the identification of dependability bottlenecks and the sensitivity analysis of the overall system to the components' dependability characteristics [1].

Based on the observation that Web services-based workflows fit into the class of systems composed of multiple operational phases characterized by potentially different requirements and goals, the paper attempts to apply methodologies and tools for dependability analysis based on the paradigm of Multiple Phased Systems (MPS, [2], [6], [7]) to this emerging category of dependability critical systems. A methodology for transforming workflow description of composite Web services into an MPS description is proposed, and, once such a description is derived, appropriate tools for MPS modeling and evaluation are applied to quantitatively assess specified dependability indicators. Hereby we use the DEEM tool to describe dependability models and to evaluate the indicators.

This paper is organized as follows. Section 2 describes the Web service flows and exposes the need for evaluating dependability indicators and discusses the related work. Section 3 introduces the MPS paradigm and the DEEM tool for dependability analysis. Section 4 discusses the possible ways of combining Web service flows as an implementation-close description of processes running in a distributed environment and MPS as a dependability modeling paradigm. Section 5 describes the model transformations performed in the VIATRA 2 framework [5] which enables a (semi-) automatic transformation of business processes (such as those built of Web services) to formal analytical models (e.g. Deterministic and Stochastic Petri Nets). Section 6 illustrates the methodology by a case study. Section 7 concludes the paper and summarizes further research directions.

2 Dependability Aspects of Web Service Flows

Present BPM tools (e.g. [26]) enable performance analysis/simulation with the restriction that all resources are available. Therefore, no faulty states can be modeled in a consistent way, failure rates and repair times cannot be considered during the analysis and no dependability analysis can be performed on the model. Similarly, there is a lack of error handling, despite the fact that some languages (for instance, Business Process Execution Language [8]) can handle exceptions. A BPEL exception handling routine, however, may contain only compensation actions which try to eliminate the effect of an uncommitted transaction, transaction time-outs, non-atomic operations etc.

The service-based approach to system integration raises the problem of defining Service Level Agreements [9] (SLA) between the provider and user of the main service. In this context, SLAs are used to describe the required quantitative parameters of a service, related to a particular client or class of clients. In general, an SLA contains measurement objectives, their guaranteed values, a measurement methodology and some goals and obligations for the participating parties. An SLA can be attached to all

service invocations, described as simple activities in a BPM, although no unified formalization of such documents exists.

As the service level of the system depends on external providers, a standardized description of the QoS parameters of Web services is needed in order to have a consistent view of the QoS at the level of composed services. Several descriptions of the QoS parameters of Web services were proposed, e.g. in [11], [12], and [13], however, no single, standardized description format was generally accepted. Accordingly, in the current paper, no specific syntax will be assumed on the service quality description, but merely only those core concepts which can be found in an arbitrary QoS definition will be referred.

To illustrate the importance of dependability analysis of Service Oriented Architecture, consider a sample process of three simple steps: receiving a request, forwarding it to an external partner (“outsourcing”) and then returning the answer to the client. The first and the last activities use internal resources (e.g. a Web server) having known performance and dependability characteristics. The second activity is however deployed on an external system, therefore its resource usage is unknown, it is described only by its Service Level Agreement parameters, for instance, “AverageResponseTime” etc. The provider of the main service has to estimate the guaranteed QoS parameters of his service, for instance the failure rate, which depends on the failure rates of the internal resources and the failure rate of the external service over which he has no control.



Fig. 1. A sample process with external method invocation

The model based analysis of a process necessitates additional information to the basic functionality of the process, such as failure rates of components, required and guaranteed response times, availability, repair times (for instance, time interval between retrying to invoke a service), etc. Non-functional design patterns such as Recovery Block should also be considered, e.g. in case of a failure, the invocation of the fastest service can be followed by calling a slower but more robust variant. The system dependability model [14] has to be created from rather different engineering models. The external service is described only by a black box model, describing the functional interface, performance and dependability-related quantitative parameters while the internal services have to be extracted from a model indicating both the functionality and the deployment to resources, extended by the non functional parameters. A uniform system-wide model has to be derived from these engineering models for the further analysis.

Available extensions of BPM lack a support of the usage of dependability parameters and fault tolerance patterns in the model. There are, however, numerous evolving standards, specifications and research in this field such as [17]. XML-based description languages such as WS-Reliability[16], WS-BaseFaults [18] can describe the

characteristics of a certain endpoint, i.e. a Web service or a port/method of a Web service. These descriptions can be associated with WSDL files [19].

As the actual description language is irrelevant from the analysis point of view, a general description language is adopted among the several emerging languages and technologies such as Web Service Level Agreements (WSLA) which contains language elements for at least a subset of the performance and dependability characteristics. As this language is quite flexible and extensible, we propose to use this for the description of non-functional parameters, as this way the external services and the internal resources could be characterized using the same description, using Service Level Agreements. A typical SLA contains the objectives to be measured, such as the transactional throughput of a web server or the response time of a remote web service, the measurement algorithm (e.g. how to compose an average measure), the fee of the service and the punishments related to violating the requirements. Guaranteed values of SLA parameters can be negotiated with the client. After such a negotiation, a complex process can be composed based on elements having well-defined QoS guarantees. The process of this negotiation is out of the scope of this paper; several ideas are discussed in [21], [22]. In our research the emphasis is on the evaluation of the process models extended with the dependability description. Hereby we suppose a WSLA-like description for the resources and the services.

Evaluation of Web Service compositions has been addressed in the literature by using Petri Net-based techniques ([29], [31]), Timed Automata [30], non-deterministic automata [32] or some kind of pi-calculus [31], [33]. PEPA models are also used to derivate quantitative characteristics of the systems and are the basis of SLA evaluation [28]. However, to our best knowledge, none of these were applied directly on a high-level (for instance, BPM) description to perform quantitative dependability analysis without the need to create a lower level model of the system, only basic verification is fully automatized.

The availability of a versatile and highly efficient tool dealing with dependability analysis of Multiple Phased Systems, combined with the appropriateness to include web-service based processes in the category of MPS systems as shown in the sequel, motivated the choices at the basis of our work.

3 Dependability Modeling: Multiple Phased Systems and DEEM

This paper elaborates on characterizing Web services-based workflows as Multiple-Phased Systems for the purpose of dependability analysis. In this section, a brief overview of MPS is provided, together with a short description of the tool DEEM for the dependability analysis of MPS.

Multiple-Phased Systems (MPS) is a class of systems whose operational life can be partitioned in a set of disjoint periods, called “phases”. During each phase, MPS execute tasks, which may be completely different from those performed within other phases. The performance and dependability requirements of MPS (such as throughput, response time, availability, etc.) can be utterly different from one phase to another. The configuration of MPS may change over time, in accordance with

performance and dependability requirements of the phase being currently executed, or simply to be more resilient to an hazardous external environment. As the so-called MPS goal may change over time, the sequence of phases of which the MPS execution is composed (the execution of a given workflow) may depend on the state (such as success/failure) of previous phases. Phased Mission Systems (PMS) and Scheduled Maintenance Systems (SMS) are two typical subtypes of MPS. Examples of MPS can be found in various application domains, such as nuclear, aerospace, telecommunications, transportation, electronics, and many other industrial fields. Because of their deployment in several critical application domains, MPS have been widely investigated, and their dependability analysis has been the object of several research studies ([2], [3], [4], [6], [7], the complete expose of the literature can be found in [2]).

Recently, a dependability modeling and evaluation tool, DEEM, specifically tailored for MPS, has been developed at the University of Florence, and ISTI-CNR [4]. DEEM relies upon Deterministic and Stochastic Petri Nets (DSPN) as the modeling formalism, and on Markov Regenerative Processes (MRGP) for the model solution [2]. When compared to existing general-purpose tools based on similar formalisms, DEEM offers advantages on both the modeling side (sub-models neatly model the phase-dependent behaviors of MPS), and on the evaluation side (a specialized algorithm allows a considerable reduction of the solution cost and time).

The rich set of modeling features provides DEEM with a two-level modeling approach in which two logically separate parts are used to represent MPS models. One is the SystemNet (SN), which represents the resource states and the failure/repair behavior of system components for each phase, and the other is the PhaseNet (PhN), which represents the execution of the various phases, as illustrated later in Figure 6. Each net is made dependent on the other one by marking-dependent predicates which modify transition rates, enabling conditions, transition probabilities, multiplicity functions, etc., to model the specific MPS features.

In DEEM, very general dependability measures for the MPS evaluation can be defined by a reward function. Among the measures assessable through such approach are the probability of successful mission completion, the relative impact of each single phase on the overall dependability figures, and the amount of useful work that can be carried out within the mission. The main motivation of using DEEM was that –as it was shown in [2]– it is a versatile and highly efficient tool for dependability modeling and evaluation of MPS systems.

4 Combining BPM as a Modeling Language and MPS as a Dependability Analysis Paradigm

As it was discussed earlier in Sect. 2., BPM models can be extended to capture non-functional parameters of the system. Therefore, an approach is needed which exploits the possibility of modeling multiple states of a resource. The modeling methodology and the evaluation procedure implemented in DEEM allow to describe the flow models of the web service systems and to analyze their dependability attributes, as shown in the next subsections.

4.1 Considering Business Process Flows as MPS

Processes in the SOA context can be considered as Multiple Phased Systems in a natural way. The two layer-representation of Multiple Phased Systems corresponds exactly to the logic of workflow-like integrated component services. The upper layer corresponds to the workflow sequence consisting of the sub-service elements while the detailed model may be used to describe the individual component services. Remind that the possibility of splitting the description into functional and non-functional aspects allows the natural expression of different parameterization of the invocation of the services and data-dependent branching in the main workflow. To illustrate the modeling issues of a business process, a more detailed view of the process in Fig.1. is presented, as shown in Figure 2.

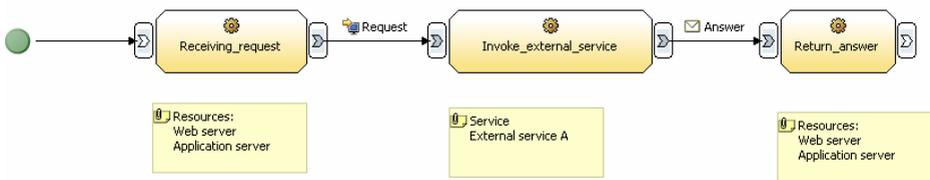


Fig. 2. Sample business process and the underlying resources

The “Receiving request” and “Return answer” activities are executed locally, i.e. on controllable and observable internal resources. Resource parameters can be modeled in a standard way using the General Resource Model UML [20] profile of the OMG. The resource faults and their effects can also be described by using the notations defined in this profile as it was shown in [10], [15]. The “Invoke external service” activity corresponds to a Web service invocation, therefore the quantitative parameters of this activity can be derived from the SLA descriptions of the service as pointed out in Sect. 2.

The dynamics of the business process flow can be treated as a Multiple Phased System in the following way. The phases (partitions of system operation) are the tasks of the BPM, unless consecutive tasks use the same internal resource. This way the context of the operation (the environment of the mission) will be different for each phase. The resource parameters –appearing in SystemNet if modeled in DEEM– such as failure rates, repair times, number of identical resources (e.g. the number of possible retransmissions of a request or the number of Web servers) may depend on the operation context, thus on the actual phase. The mission goal can change over the time and the execution of the process (i.e. the mission goal) may depend on the result of previous phases and the system state. For instance, if validating a credit card does not terminate within a predefined timeout, then a flight ticket reservation cannot be confirmed, but is saved as a conditional reservation.

The chosen method to evaluate the dependability of the web service systems is describing the model in BPM and transforming it to a MPS model, since the basic description language –in which the process is built– is easy to use, a wide range of tools are available, and an implementation skeleton (i.e. the workflow control description) can be generated directly from the model after passing the dependability

analysis. Moreover, BPM activities can be converted into phases of a MPS in a quite natural way while the opposite direction (i.e. generating the skeleton of a control flow from a MPS model) raises several questions as many activities can be described in one phase, if their dependability parameters are the same. Using model transformations instead of describing the model in a meta-language brings the benefit of easy implementation of additional transformations (for instance, model checking based on qualitative properties of the mode) as the model is stored in the format of the graph transformation tool. The transformation tool also enables the generation of practically any type of output (which can be further the basis of a runtime validation).

The basic BPM model should be extended with some information about the required behavior of components (basic activities), such as maximum response time, maximum number of timeouts in a given time interval, guaranteed rate of good answers for a prefixed number of requests, availability, etc. All these characteristics can be derived from the characteristics of the resources and the software implementation (if known) in the case of internal services (those we have control over) or from the Service Level Agreements in the case of external service. The business process model can be transformed to a MPS according to the following rules:

- The different activities will be different phases in PhaseNet with different goals, dependability metrics and resources.
- The performance and dependability characteristics of resources and services will determine the SystemNet parameters such as transition rate, initial marking, etc.
- The dependencies between PhaseNet and SystemNet are given by the task-resource bindings and SLAs of the (both internal and external) services.
- The measurements of a MPS analysis are determined by the “business measures” of the BPM, i.e. the QoS parameters of the main service.

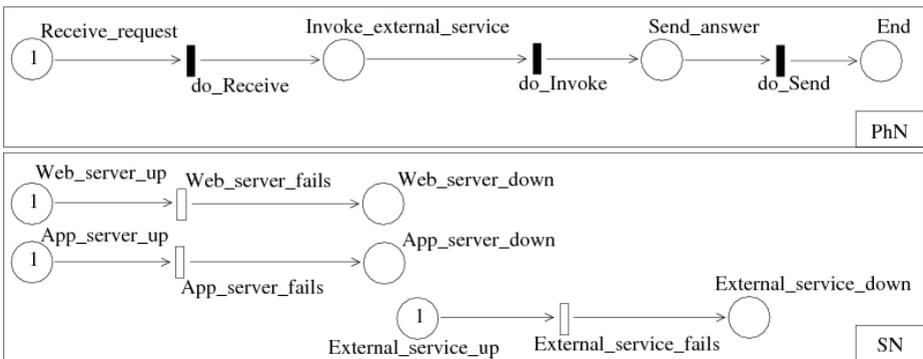


Fig. 3. The sample process seen as a MPS system and modeled in DEEM

Figure 3 shows the sample process as a Multiple Phased System. Note that the parameters of the DSPN are derived from the BPM parameters based on the resource descriptions and SLAs. The expected durations of the timed transitions of the PhaseNet correspond to the estimated execution time for activities of the business process. For instance, the expected duration of the transition “do_Receive” results from the average execution time of “Receive request” activity while the expected duration of

the transition “do_Invoke” is derived from the average response time of the external service, described in the corresponding SLA. Error manifestation is expected to happen at the invocation of an operation using a resource. Resource faults inducing errors are modeled by the timed transitions of the SystemNet while the enabling conditions of these transitions model the resource allocation.

For instance, the transition “Web server fails” is enabled during the phases which correspond to tasks using the web server while the transition rate corresponds to the expected failure rate of a server during a typical transaction. These parameters are represented in the IBM WBI as the “description” of the resources (since the definition of failure rates of resources is not supported by the present BPM tool). For external services, such as “External service” in this example, the failure rate is derived from the Service Level Agreement, also using a textual field in the modeling tool. The main difference between the internal resource usage and the external service invocation is that in the former case, multiple resources can be used simultaneously and a fault in any of them prohibits the proper service while in the latter case only a single service and –at most– one resource, namely the application server is used.

5 Model Transformations with VIATRA2

As it was mentioned in Sect. 4., the dependability indicators of business processes can be evaluated if they are transformed into a MPS model. The model transformation-based analysis of business process descriptions consists of the steps shown in Figure 4.

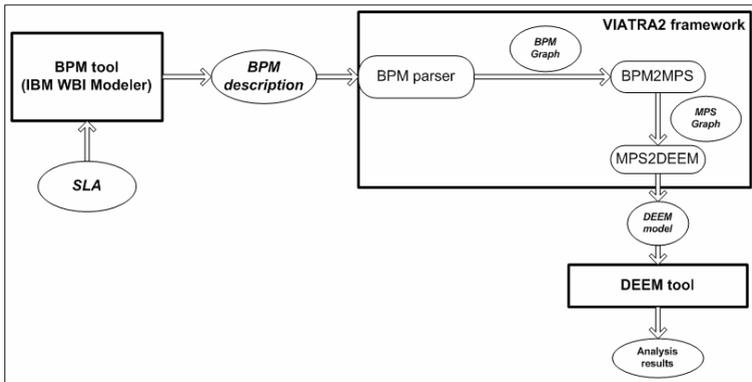


Fig. 4. Transformation of business processes

Using model transformations for the analysis of high-level system models is part of the Model Driven Architecture (MDA) concept. The motivation for using model transformations in the VIATRA (Visual Automated Model Transformations) framework was the *extensibility* of the transformation engine by additional parsers and plugins which enable the decoupling of the format of the source model and the target analysis platform.

First, the engineering model of the functionality –enriched by dependability parameters– is taken to be analyzed by formal methods. This model will be generated by

a BPM tool [26] and transformed by the VIATRA2 framework into a DEEM model. To build a mathematical model from the high level business description in an automated way, the BPM is parsed into a graph representation (“BPM Graph”). In our case, this will be the inner representation language of the VIATRA2 tool, which is a public domain model transformation framework, developed at BUTE [23]. It is now part of the Eclipse GMT project [5]. In the VIATRA2, graph pattern matching [24] is controlled by Abstract State Machines [25].

Then, graph transformations are performed upon this parsed model in order to generate a graph which represents the relevant elements of the system in the target paradigm (“MPS Graph”). In this case, the target paradigm is MPS and the target model representation format is that of the DEEM tool. However, as it will be discussed later, the transformation itself was implemented in two steps. Once the model can be read by the target analysis tool, a precise analysis method (in this case, the Markov Regenerative Process-based dependability evaluation) can be performed.

The transformations were implemented with the VIATRA2 model transformation framework. The automatic generation of a DEEM model consists of three basic steps:

1. Importing the XML files which contain the description of different aspects of the BPM model (“BPM description”), such as the basic process model and the actual values of the variables which determine the runtime behavior of the system, e.g. the probabilities of paths to be followed after decisions. This step was implemented using the built-in BPM parser component of the VIATRA2 framework, which creates an inner graph representation –in the VPML language of the tool– of the business process (“BPM Graph”).
2. Transforming the graph representation of the BPM structures and concepts into a graph representation of a multiple phased system. The model transformation (“bpm2mps”) itself is implemented in this step. The metamodel of a general MPS description contains the elements of a DSPN-based representation of MPS, such as the places, the transitions and the arcs of the SystemNet and the PhaseNet (“MPS Graph”). The transformation is described by precondition patterns matching to the concepts of the BPM metamodel and the corresponding postcondition patterns give the equivalent Petri Net structures, describing a Phase Mission System still in the graph representation language. As this transformation is based on a generic MPS metamodel, the analysis tool can be replaced by another Petri Net based tool without any change in this transformation.
3. Code generation: once a graph representation of the MPS is available, the text file in the DEEM format can be generated by a simple transformation (“mps2deem”). This transformation is designed to take a graph, in which the elements are stored in a tree structure and references between them describe the logical connections, and generate a text file (“DEEM model”). The main reason of the separation of model transformation and the code generation is twofold; first, this way the changes in the tool representation format (or even the replacement of the DEEM by another analysis tool) can be easily tracked and do not interfere with algorithm of the transformation of the main concepts. Therefore, the transformations are maintainable. Second, since the DEEM representation is a flat format, the whole graph tree is needed for the code generation, and therefore this step cannot be started before the entire model transformation is finished.

In order to analyze business processes with DEEM (“Analysis results”), the two transformations (“bpm2mps” and “mps2deem”) were implemented in the VIATRA2 framework. At the moment there are some limitations on BPM elements due to the BPM parser of the framework, but anyway they do not affect the essence of the methodology.

6 Case Study

Consider an insurance company with a database containing client data (e.g. previous accidents) wanting to provide a premium calculator service which receives client data and returns an estimated insurance fee for the given person. Consider that this company wants to interact with other companies to complete its knowledge about a client’s insurance record. This interaction is done via Web service interface; the partners provide similar premium calculator services.

The clients of such an application are employees of the company, individual brokers, other companies, registered users, etc. The company implements this functionality as a Web service to support communication between heterogeneous, loosely-coupled systems. The company wants to assure QoS parameters for the clients. Therefore, its own resources and services have to match several expectations just as the external services.

There are different types of clients with different QoS requirements against the premium calculator service. For instance, a client with a “Golden value” contract (another insurance company) may have different expectations against the system than a registered user accessing the service from a home PC.

Requests which mean a big risk (a calculation for an insurance of big amount or for a client with missing personal data) have to be checked by other partners to eliminate the chance of failure or cheating. On the other hand, different partner companies offer their calculator services (which are external activities in the process flow) for different prices.

The measures of interest are –among others– the following:

- The probability that a client request fails (for different types of clients).
- Performability metrics which show the cost of dependability, i.e. which external services to invoke at given QoS parameters and price. Requests of different client types, of course, can be forwarded to different external partners in order to assure the required QoS at a reasonable price.

Finally, sensitivity analysis is required to evaluate the effect of component failure rates on above measures.

6.1 The Example Model in BPM

This section describes the high-level BPM model of the example. The concrete modeling tool is IBM WBI Modeler which, on the one hand, supports the modeling of resources (in this case, the quantitative parameters of the services) and, on the other hand, has a BPEL export feature.

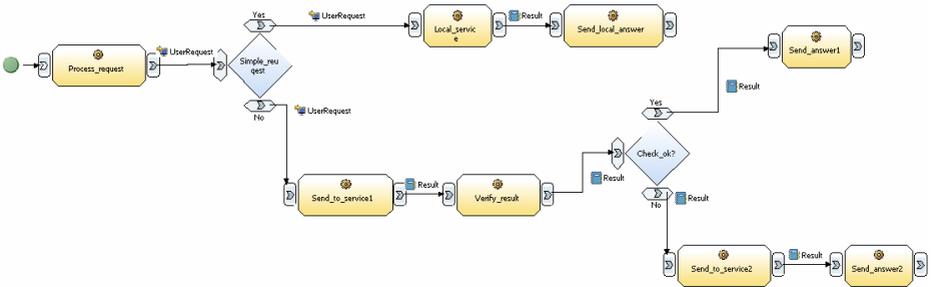


Fig. 5. The example in IBM WBI modeler

The rounded rectangles represent the internal and external activities. Parameters of the resources and the services are stored in the model repository but are not visualized. To comply with the BPEL standard, internal activities can also be accessed via a SOAP interface, but their QoS parameters depend on resources with known characteristics.

6.2 The Example Model as an MPS

This section describes the MPS model of the example business flow, showing the general method of transforming a BPM to a MPS. The system can be considered as a MPS in the following way.

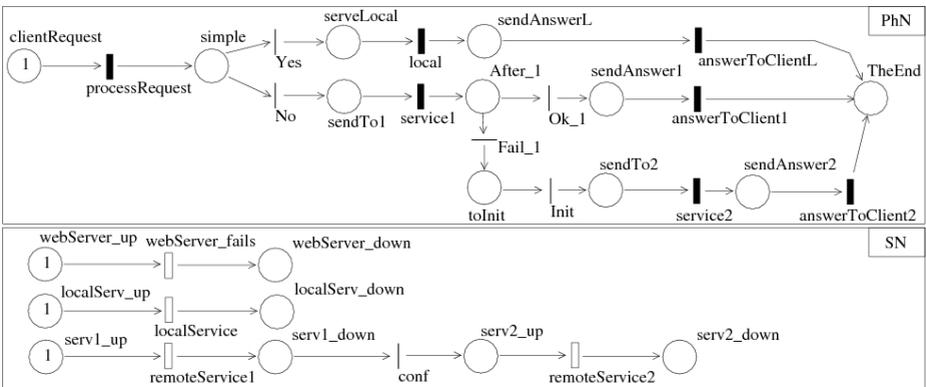


Fig. 6. The MPS model of the example

The Web server is modeled in the SystemNet (the lower partition of Fig. 6) which can be reconfigured according to that actual phase. The Web server can fail with a failure rate which is the parameter of the “webServer_fails” transition. This motivates the usage of the same topology (places and transitions) for representing all resources; the transition rates may change over time, according to the actual phase.

The first phase of the system is receiving a message from the client. This corresponds to the first activity of the business process. The second phase is upon a decision; whether the request can be served locally (in this case, no remote web service is invoked) or it needs to be sent to a remote partner. In the latter case, the answer is verified against some basic requirements, for instance, the presence and the consistency of all required data fields are checked.

If additional information is needed, the request is sent to a backup service, provided by another partner, and built upon another database. The first activity of the workflow (“Process request”) is the only element of a sequence of internal service invocations. Therefore, it is the first phase of the system. The length of the phase will be represented by a timed transition (processRequest in Fig. 6) with a transition duration determined by the length of the task in the business process. During this phase, the system can fail if the internal resource, in this case the Web server, crashes.

The next phase is selected according to the user type; the simple calculations are served locally while the difficult calculations, e.g. those of users with missing data or big value of insurance, are sent to external partners. In this version, the “Recovery Block” pattern is implemented in a service oriented environment, which could be called “Recovery Block-like Service Invocations”. This means the invocation of a primary service, and if the answer is not acceptable –for instance, some user record is empty– then the invocation of a backup service.

The parameters of external service invocations (modeled by phase “service1” and “service2”) are determined by the SLAs. The failure rate of the services comes from the UpTimeRatio parameter from the SLA (which is represented in the BPM tool as a resource parameter of the services which represent the remote partners). The possible reconfiguration of the system, i.e. the resending of the request to the “backup service”, is represented by the transition “conf” in the SystemNet.

6.3 Dependability Analysis Results

To illustrate how dependability analysis constitutes a support to the provider of the composed service, we answer the questions “What is the probability of the failure of a client request?” and “Which is the most appropriate external service provider from the set of available providers?”.

In a real scenario, the parameters of the services and resources are described by Service Level Agreements. As our aim is to present a methodology for the evaluation of dependability indicators, we used some sample values based on a measurement performed against public domain web services, such as the web service interface of google [27]. Hereby we suppose ten available service alternatives for Service1, which have their parameters described in Table 1 (considering the same response time). Please remember that these do not have realistic meanings, but have been chosen just to illustrate the possibilities of such an analysis. Due to the space problems, we do not include a table with the fix parameters used for the evaluations (e.g., the costs in the reward measure, the duration of the phases, etc.).

Table 1. Parameters of services in the example

Service alternatives	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
Failure rate	0.010	0.012	0.017	0.020	0.025	0.030	0.040	0.042	0.048	0.070
Service price	14.0	13.5	13.3	13.0	12.8	12.6	12.5	12.2	12.0	11.5

The aim of this analysis is at determining the impact of the price and dependability characteristics of an external web service on the probability of the failure of a client request and on the income of the composite service. This income is the fee that a client pays for the service minus the sum of the prices of the invoked local and external services. As the client receives a compensation for every failed request, this value has to be considered as a penalty. The measures of interest are defined in DEEM as

```

probServiceFail = IF ( MARK(webServer_down)=1 OR
MARK(localServ_down)=1 OR MARK(serv1_down)=1 OR
MARK(serv2_down)=1 ) THEN (1) ELSE (0) //failure prob

ServiceReward = [VAR(ClientFee) -
VAR(serv1Price)*FUN(serv1Succ) -
VAR(serv2Price)*FUN(serv2Succ) -
VAR(localPrice)*FUN(localServSucc)] *
(1-FUN(serviceFail)) -
FUN(serviceFail)*VAR(servicePenalty) //service reward
    
```

The results of the analysis is shown in Fig. 7.

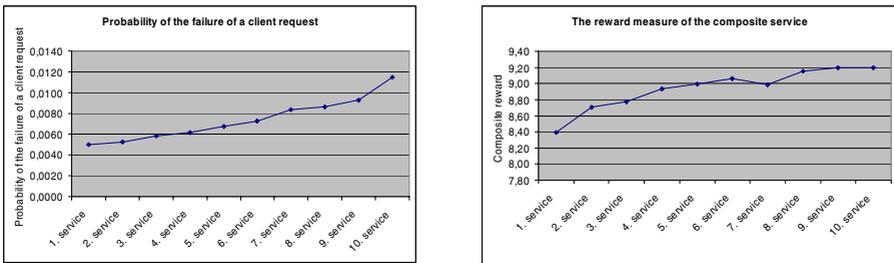


Fig. 7. Results of the dependability analysis

Based on dependability aspects only, 1.service is obviously the best choice as the number of failed requests has a minimum for this service. However, if the services are evaluated against the performability measure, then 10.service seems to be optimal as it has the highest reward value. If both aspects are considered, 9.service should be chosen instead, as it has almost as good performability measure as that of 10.service with a significantly lower probability of failure of a client request.

7 Conclusions

We presented a methodology for transforming higher level models of Service Oriented Architectures into a formal description in order to perform dependability analysis. Based on the observation that Web services-based workflows fit within the Multiple Phased Systems, model transformations were implemented in the VIATRA2 framework to perform precise mathematical analysis. Business process descriptions extended with quantitative parameters taken from SLAs were transformed, by using semi-automated transformations, into a precise mathematical model, a formal description of a Multiple Phased System, which can be solved by the dependability evaluation tool DEEM.

The current research direction is to extend the proposed methodology to analyze high level models described in BPEL and XML-based Web service description languages to provide a dependability analysis for a wider toolset. This way, a really platform (and vendor) independent analysis framework can be established. SLA-driven synthesis of web service compositions is another important research direction.

References

1. M. Martinello. Availability modeling and evaluation of web-based services –A pragmatic approach. PhD Thesis. LAAS-CNRS, 2005.
2. I. Mura, A. Bondavalli, X. Zang, and K. S. Trivedi, “Dependability modelling and evaluation of phased mission systems: a DSPN approach,” in IEEE DCCA-7 - 7th IFIP Int. Conference on Dependable Computing for Critical Applications, San Jose, CA, USA, 1999, pp. 299–318.
3. I. Mura and A. Bondavalli, “Markov regenerative stochastic Petri nets to model and evaluate phased mission systems dependability,” IEEE Transactions on Computers, vol. 50, no. 12, pp. 1337–1351, 2001.
4. A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and I. Mura. Dependability modeling and evaluation of multiple-phased systems, using DEEM. IEEE Transactions on Reliability, 53(4):509-522, 2004.
5. The VIATRA2 Model Transformation Framework, Generative Model Transformer Project, The Eclipse Foundation. <http://eclipse.org/gmt/>
6. M. Smotherman and K. Zemoudeh, “A non-homogeneous Markov model for phased-mission reliability analysis,” IEEE Transactions on Reliability, vol. 38, no. 5, pp. 585–590, 1989.
7. M. Alam and U. M. Al-Saggaf, “Quantitative reliability evaluation of repairable phased-mission systems using Markov approach,” IEEE Transactions on Reliability, vol. R-35, no. 5, pp. 498–503, 1986.
8. Specification: Business Process Execution Language for Web Services Version 1.1, May 2003. <http://www-128.ibm.com/developerworks/library/ws-bpel/>
9. D. Menasce, V. A. F. Almeida. Capacity Planning for Web Services: Metrics, Models, and Methods. Prentice Hall, 2001.
10. I. Majzik, A. Pataricza, and A. Bondavalli. Stochastic dependability analysis of system architecture based on UML models. In R. De Lemos, C. Gacek, and A. Romanovsky, editors, Architecting Dependable Systems, LNCS 2677, pp. 219-244. Springer-Verlag, Berlin, Heidelberg, New York, 2003.

11. Web Service Level Agreements Project. <http://www.research.ibm.com/wsla/>
12. Web Services Flow Language (WSFL 1.0) - Appendix C: Endpoint Property Extensibility Elements. IBM Software Group, 2001.
13. V. Tosic, B. Paguerk, K.Patel. WSOL – A Language for the Formal Specification of Various Constraints and Classes of Service for Web Services. Research Report, Carleton University, 2002
14. A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing IEEE Transactions on Dependable and Secure Computing, Vol.1, N.1, pp.11-33, 2004
15. A. Pataricza: From the General Resource Model to a General Fault Modeling Paradigm? Workshop on Critical Systems Development with UML at UML 2002, Dresden, Germany
16. WS–Reliability. OASIS Standard
http://docs.oasis-open.org/wsrn/ws-reliability/v1.1/wsrn-ws_reliability-1.1-spec-os.pdf
17. A. Graziano, S. Russo, V. Vecchio, P. Foster, Metadata models for QoS-aware information management systems. In Proc. of SEKE 2002, Ischia, Italy, 2002.
18. Web Services Base Faults. Oasis.
<http://docs.oasis-open.org/wsrn/2004/06/wsrn-WS-BaseFaults-1.2-draft-02.pdf>
19. Web Service Description Language 1.1. W3C.org. <http://www.w3.org/TR/wsdl>
20. OMG Group, General Resource Model (GRM), URL: <http://www.omg.com>
21. L. Zeng, B. Benatallah, M. Dumas. Quality Driven Web Services Composition. In Proceedings of WWW2003, May 20-24, 2003, Budapest, Hungary.
22. S. Ran. A model for web services discovery with QoS. ACM SIGecom Exchanges, Vol. 4., N.1, pp. 1-10. ACM Press, 2003.
23. D. Varró, G. Varró, A. Pataricza, “Designing the Automatic Transformation of Visual Languages,” Science of Computer Programming, 44:205-227 2002.
24. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg (eds.). Handbook on Graph Grammars and Computing by Graph Transformation, vol. 2: Applications, Languages and Tools, World Scientific, 1999.
25. E. Börger and R. Stark. Abstract State Machines. A method for High-Level System Design and Analysis. Springer-Verlag, 2003.
26. IBM Corporation. WebSphere Business Integrator 5.1
<http://www-06.ibm.com/software/integration/>
27. Google Web API (beta). <http://www.google.com/apis/index.html>
28. J.T. Bradley, N.J. Dingle, S.T. Gilmore, W.J. Knottenbelt. Derivation of Passage-time Densities in PEPA Models using ipc: the Imperial PEPA Compiler. In Proc. of MASCOTS'03, Orlando, USA, 2003, pp. 344-351.
29. C. Ouyang, E. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede WofBPEL: A Tool for Automated Analysis of BPEL Processes. In Proc. of ICSC 2005, Amsterdam, The Netherlands. 2005. pp. 484-489.
30. R. Kazhamiakin, P. Pandya, M. Pistore. Modelling and Analysis of Time-related Properties in Web Service Compositions. In Proc. of WESC'05, Amsterdam, The Netherlands, 2005.
31. W. M.P. van der Aalst, M. Dumas, A.H.M ter Hofstede, N. Russell, P. Wohed, H. M. W. Verbeek. Life After BPEL?. In Proc. of WS-FM, Versailles, France. 2005. pp 35-50.
32. J. Koehler, G. Tirenni, S. Kumaran, From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods, EDOC, Lausanne, Switzerland, 2002, pp. 96-106.
33. R. Milner. Communicating and Mobile Systems: The Pi-Calculus. Cambridge University Press, Cambridge, UK, 1999.

Empirical Analysis and Statistical Modeling of Attack Processes based on Honeypots*

M. Kaâniche¹, E. Alata¹, V. Nicomette¹, Y. Deswarte¹, M. Dacier²

¹LAAS-CNRS, ²Eurecom

Abstract

Honeypots are more and more used to collect data on malicious activities on the Internet and to better understand the strategies and techniques used by attackers to compromise target systems. Analysis and modeling methodologies are needed to support the characterization of attack processes based on the data collected from the honeypots. This paper presents some empirical analyses based on the data collected from the Leurré.com honeypot platforms deployed on the Internet and presents some preliminary modeling studies aimed at fulfilling such objectives.

1. Introduction

Several initiatives have been developed during the last decade to monitor malicious threats and activities on the Internet, including viruses, worms, denial of service attacks, etc. Among them, we can mention the Internet Motion Sensor project [1], CAIDA [2], DShield [3], and CADHo [4]. These projects provide valuable information on security threats and the potential damage that they might cause to Internet users. Analysis and modeling methodologies are necessary to extract the most relevant information from the large set of data collected from such monitoring activities that can be useful for system security administrators and designers to support decision making. The designers are mainly interested in having representative and realistic assumptions about the kind of threats and vulnerabilities that their system will have to cope with once it is used in operation. Knowing who are the enemies and how they proceed to defeat the security of target systems is an important step to be able to build systems that can be resilient with respect to the corresponding threats. From the system security administrators' perspective, the collected data should be used to support the development of efficient early warning and intrusion detection systems that will enable them to better react to the attacks targeting their systems.

As of today, there is still a lack of methodologies and significant results to fulfill the objectives described above, although some progress has been achieved recently in this field. The CADHo project "Collection and

* This paper was published in the Supplemental Proceedings of the 2006 IEEE International Conference on Dependable Systems and Networks (DSN 2006), Workshop on Empirical Evaluation of Dependability and Security (WEEDS), Philadelphia, USA, pp. 119-124, October 18-20, 2006.

Analysis of Data from Honeypots” [4], an ongoing research action started in September 2004, is aimed at contributing to filling such a gap by carrying out the following activities:

- deploying a distributed platform of honeypots [5] that gathers data suitable to analyze the attack processes targeting a large number of machines connected to the Internet;
- developing analysis methodologies and modeling approaches to validate the usefulness of this platform by carrying out various analyses, based on the collected data, to characterize the observed attacks and model their impact on security.

A honeypot is a machine connected to a network but that no one is supposed to use. In theory, no connection to or from that machine should be observed. If a connection occurs, it must be, at best an accidental error or, more likely, an attempt to attack the machine.

The *Leurré.com* data collection environment [5], set up in the context of the CADHo project, has deployed, as of to date, thirty five honeypot platforms at various locations from academia and industry, in twenty five countries over the five continents. Several analyses carried out based on the data collected so far from these honeypots have revealed that very interesting observations and conclusions can be derived with respect to the attack activities observed on the Internet [4, 6-9]. In addition, several automatic data analyses and clustering techniques have been developed to facilitate the extraction of relevant information from the collected data. A list of papers detailing the methodologies used and the results of these analyses is available in [6].

This paper focuses on modeling-related activities based on the data collected from the honeypots. We first discuss the objectives of such activities and the challenges that need to be addressed. Then we present some examples of models obtained from the data.

The paper is organized as follows. Section 2 presents the data collection environment. Section 3 focuses on the modeling of attacks based on the data collected from the honeypots deployed. Modeling examples are presented in Section 4. Finally, Section 5 discusses future work.

2. The data collection environment

The data collection environment (called *Leurré.com* [5]) deployed in the context of the CADHo project is based on low-interaction honeypots using the freely available software called *honeyd* [10]. Since September 2004, 35 honeypot platforms have been progressively deployed on the Internet at various geographical locations. Each platform emulates three computers running Linux RedHat, Windows 98 and Windows NT, respectively, and various services such as ftp, web, etc. A firewall ensures that connections cannot be initiated from the computers, only replies to external solicitations are allowed. All the honeypot platforms are centrally managed to ensure that they have exactly the same configuration. The data gathered by each platform are securely uploaded to a centralized database with the complete content, including payload of all packets sent to or from these honeypots, and additional information to facilitate its analysis, such as the IP geographical localization of packets’ source addresses, the OS of the attacking machine, the local time of the source, etc.

3. Modeling objectives

Modeling involves three main steps:

- 1) The definition of the objectives of the modeling activities and the quantitative measures to be evaluated.

- 2) The development of one (or several) models that are suitable to achieve the specified objectives.
- 3) The processing of the models and the analysis of the results to support system design or operation activities.

The data collected from the honeypots can be processed in various ways to characterize the attack processes and perform predictive analyses. In particular, modeling activities can be used to:

- Identify the probability distributions that best characterize the occurrence of attacks and their propagation through the Internet.
- Analyze whether the data collected from different platforms exhibit similar or different malicious attack activities.
- Model the time relationships that may exist between attacks coming from different sources (or to different destinations).
- Predict the occurrence of new waves of attacks on a given platform based on the history of attacks observed on this platform as well as on the other platforms.

For the sake of illustration, we present in the following sections simple preliminary models based on the data collected from our honeypots that are aimed at fulfilling such objectives.

4. Examples

The examples presented in the following address:

- 1) The analysis of the time evolution of the number of attacks taking into account the geographic location of the attacking machine.
- 2) The characterization and statistical modeling of the times between attacks.
- 3) The analysis of the propagation of attacks throughout the honeypot platforms.

The data considered for the examples has been collected from January 1st, 2004 to April 17, 2005, corresponding to a data collection period of 320 days. We take into account the attacks observed on 14 honeypot platforms among those deployed so far. The selected honeypots correspond to those that have been active for almost the whole considered period. The total number of attacks observed on these honeypots is 816476. These attacks are not uniformly distributed among the platforms. In particular, the data collected from three platforms represent more than fifty percent of the total attack activity.

4.1. Attack occurrence and geographic distribution

The preliminary models presented in this sub-section address: i) the time-evolution modeling of the number of attacks observed on different honeypot platforms, and ii) the analysis of potential correlations for the attack processes observed on the different platforms taking into account the geographic location of the attacking machines and the proportion of attacks observed on each platform, wrt. the global attack activity.

Let us denote by:

- $Y(t)$ the function describing the evolution of the number of attacks per unit of time observed on all the honeypots during the observation period,

- $X_j(t)$ the function describing the evolution of the number of attacks per unit of time observed on all the honeypots during the observation period for which the IP address of the attacking machine is located in country j .

In a first stage, we have plotted, for various time periods, $Y(t)$ and the curves $X_j(t)$ corresponding to different countries j . Visual inspection showed surprising similarities between $Y(t)$ and some $X_j(t)$. To confirm such empirical observations, we have then decided to rigorously analyze this phenomenon using mathematical linear regression models.

Considering a linear regression model, we have investigated if $Y(t)$ can be estimated from the combination of the attacks described by $X_j(t)$, taking into account a limited number of countries j . Let us denote by $Y^*(t)$ the estimated model.

Formally, $Y^*(t)$ is defined as follows:

$$Y^*(t) = \sum \alpha_j X_j(t) + \beta \quad j= 1, 2, .. k \quad (1)$$

Constants α_j and β correspond to the parameters of the linear model that provide the best fit with the observed data, and k is the number of countries considered in the regression.

The quality of fit of the model is measured by the statistics R^2 defined by:

$$R^2 = \frac{\sum (Y^*(i) - Y_{av})^2}{\sum (Y(i) - Y_{av})^2} \quad (2)$$

$Y(i)$ and $Y^*(i)$ correspond to the observed and estimated number of attacks for unit of time i , respectively. Y_{av} is the average number of attacks per unit of time, taking into account the whole observation period.

Indeed, R is the correlation factor between the estimated model and the observed values. The closer the R^2 value is to 1, the better the estimated model fits the collected data.

We have applied this model considering linear regressions involving one, two or more countries. Surprisingly, the results reveal that a good fit can be obtained by considering the attacks from one country only. For example, the models providing the best fit taking into account the total number of attacks from all the platforms are obtained by considering the attacks issued from either UK, USA, Russia or Germany only. The corresponding R^2 values are of the same order of magnitude (0.944 for UK, 0.939 for USA, 0.930 for Russia and 0.920 for Germany), denoting a very good fit of the estimated models to the collected data. For example, the estimated model obtained when considering the attacks from Russia only is defined by equation (3):

$$Y^*(t) = 44.568 X_j(t) + 1555.67 \quad (3)$$

$X_j(t)$ represents the evolution of the number of attacks from Russia. Figure 1 plots the evolution of the observed and estimated number of attacks per unit of time during the data collection period considered in this example. The unit of time corresponds to 4 days. It is noteworthy that, similar conclusions are obtained if we consider another granularity for the unit of time, for example one day, or one week.

These results are even more surprising that the attacks from Russia and UK represent only a small proportion of the total number of attacks (1.9% and 3.7% respectively). Concerning the USA, although the proportion is higher (about 18%), it is not sufficient to explain the linear model.

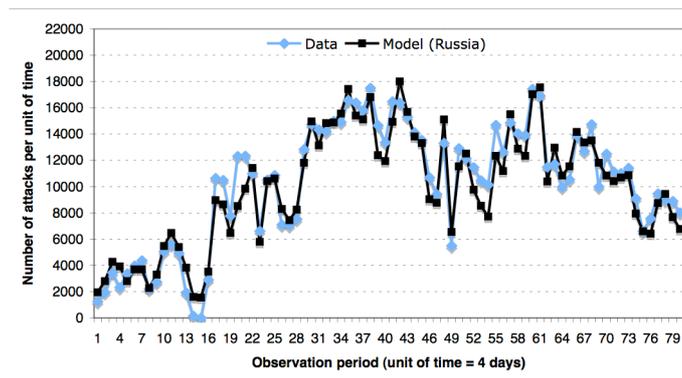


Figure 1- Evolution of the number of attacks per unit of time observed on all the platforms and estimated model considering attacks from Russia only

We have applied similar analyses by respectively considering each honeypot platform in order to investigate if similar conclusions can be derived by comparing their attack activities per source country to their global attack activities. The results are summarized in Table 1. The second column identifies the source country that provides the best fit. The corresponding R^2 value is given in the third column. Finally, the last three columns give the R^2 values obtained when considering UK, USA, or Russia in the regression model.

It can be noticed that the quality of the regressions measured when considering attacks from Russia only is generally low for all platforms (R^2 less than 0.80). This indicates that the property observed at the global level is not visible when looking at the local activities observed on each platform. However, for the majority of the platforms, the best regression models often involve one of the three following countries: USA, Germany or UK, which also provide the best regressions when analyzing the global attack activity considering all the platforms together. Two exceptions are found with P6 and P8 for which the observed attack activities exhibit different characteristics with respect to the origin of the attacks (Taiwan, China), compared to the other platforms.

Platform	Country providing the best model	R^2 Best model	R^2 UK	R^2 USA	R^2 Russia
P1	Germany	0.895	0.873	0.858	0.687
P2	USA	0.733	0.464	0.733	0.260
P4	Germany	0.722	0.197	0.373	0.161
P5	Germany	0.874	0.869	0.872	0.608
P6	UK	0.861	0.861	0.699	0.656
P8	Taiwan	0.796	0.249	0.425	0.212
P9	Germany	0.754	0.630	0.624	0.631
P11	China	0.746	0.303	0.664	0.097
P13	Germany	0.738	0.574	0.412	0.389
P14	Germany	0.708	0.510	0.546	0.087
P20	USA	0.912	0.787	0.912	0.774
P21	SPAIN	0.791	0.620	0.727	0.720
P22	USA	0.870	0.176	0.870	0.111
P23	USA	0.874	0.659	0.874	0.517
Global	UK	0.944	0.944	0.939	0.930

Table 1 – Estimated models for each platform: correlation factors for the countries providing the best fit and for UK, USA and Russia

The trends discussed above have been also observed when considering a different granularity for the unit of time (e.g., 1 day or 1 week) as well as different data observation periods.

To summarize, two main findings can be derived from the results presented above:

- Some trends exhibited at the global level considering the attack processes on all the platforms together are not observed when analyzing each platform individually (this is the case, for example, of attacks from Russia). On the other hand, we have observed the other situation where the trends observed globally are also visible locally on the majority of the platforms (this is the case, for example, of attacks from USA, UK and Germany).
- The attack processes observed on each platform are very often highly correlated with the attack processes originating from a particular country. The country providing the best regressions locally, does not necessary exhibit high correlations when considering other platforms or at the global level. These trends seem to result from specific factors that govern the attack processes observed on each platform.

4.2. Distribution of times between attacks

In this example, we focus on the analysis and the modeling of the times between attacks observed on different honeypot platforms.

Let us denote by t_i , the time separating the occurrence of attack i and attack $(i-1)$. Each attack is associated to an IP address, and its occurrence time is defined by the time when the first packet is received from the corresponding address at one of the three virtual machines of the honeypot platform. All the packets received from the same IP address within 24 hours are supposed to belong to the same attack session.

We have analyzed the distribution of the times between attacks observed on each honeypot platform. Our objective was to find analytical models that faithfully reflect the empirical data collected from each platform. In the following, we summarize the results obtained considering 5 platforms for which we have observed the highest attack activity.

4.2.1. Empirical analyses

Table 2 gives the number of intervals of times between attacks observed at each platform considered in the analysis as well as the corresponding number of IP addresses. As illustrated by Figure 2, most of these addresses have been observed only once at a given platform. Nevertheless, some IP addresses have been observed several times, the maximum number of visits per IP address for the five platforms was 57, 96, 148, 183, and 83 (respectively). Indeed, the curves plotting the number of IP addresses as a function of the number of attacks for each address follow a heavy-tailed power law distribution. It is noteworthy that such distributions have been observed in many performance and dependability related studies in the context of the Internet, e.g., transfer and interarrival times, burst sizes, sizes of files transferred over the web, error rates in web servers, etc.

	P5	P6	P9	P20	P23
Number of t_i	85890	148942	46268	224917	51580
Number of IP addresses	79549	90620	42230	162156	47859

Table 2 – Numbers of intervals of times between attacks (t_i) and of different IP addresses at each platform

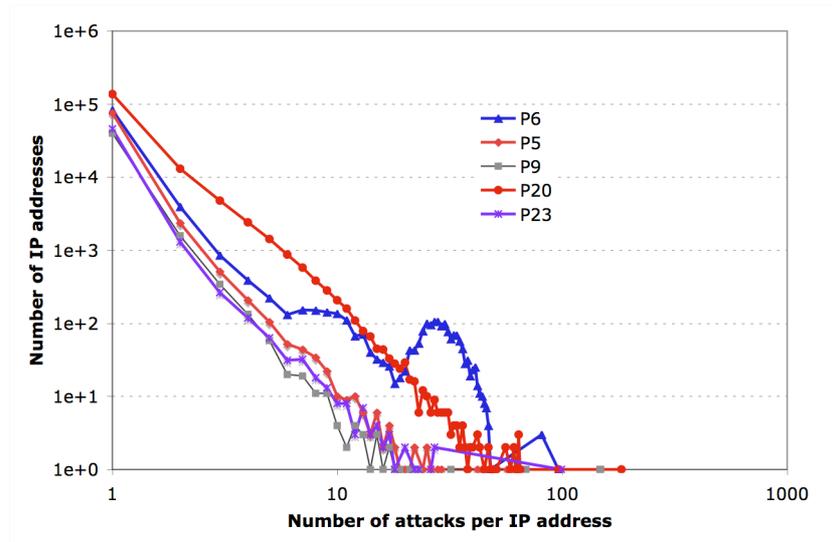


Figure 2- Number of IP addresses versus the number of attacks per IP address at each platform (log-log scale)

4.2.2. Modeling

Finding tractable analytical models that faithfully reflect the observed times between attacks is useful to characterize the observed attack processes and to find appropriate indicators that can be used for prediction purposes. We have investigated several candidate distributions, including Weibull, Lognormal, Pareto, and the Exponential distribution, which are traditionally used in reliability related studies. The best fit for each platform has been obtained using a mixture model combining a Pareto and an exponential distribution.

Let us denote by T the random variable corresponding to the time between the occurrence of two consecutive attacks at a given platform, and t a realization of T . Assuming that the probability density function $pdf(t)$ associated to T is characterized by a mixture distribution combining a Pareto distribution and an exponential distribution, then $pdf(t)$ is defined as follows.

$$pdf(t) = P_a \frac{k}{(t+1)^{k+1}} + (1 - P_a) \lambda e^{-\lambda t}$$

k is the index parameter of the Pareto distribution, λ is the rate associated to the exponential distribution and P_a is a probability.

We have used the R statistical package [11] to estimate the parameters that provide the best fit to the collected data. The quality of fit is assessed by applying the Kolmogorov-Smirnov statistical test. The results are presented in Figure 3. It can be noticed that for all the platforms, the mixed distribution provides a good fit to the observed data whereas the exponential distribution is not suitable to describe the observed attack processes. Thus, the traditional assumption considered in hardware reliability evaluation studies assuming that failures occur according to a Poisson process does not seem to be satisfactory when considering the data observed from our honeypots. These results have been also confirmed when considering the data collected during other observation periods.

4.3. Propagation of attacks

Besides analyzing the attack activities observed at each platform in isolation, it is useful to identify phenomena that reflect propagation of attacks through different platforms. In this section, we analyze simple scenarios where a propagation between two platforms is assumed to occur when the IP address of an attacking machine observed at a given platform is also observed at another platform. Such a situation might occur for example as a result of a scanning activity or might be resulting from the propagation of worms.

For the sake of illustration, we restrict the analysis to the five platforms considered in the previous example. For each attacking IP address in the data collected from the five platforms during the period of the study, we identified: 1) all the occurrences with the same source address, 2) the times of each occurrence and 3) the platform on which each occurrence has been reported. A *propagation* is said to occur for this IP address from platform P_i to platform P_j when the next occurrence of this address is observed on P_j after visiting P_i .

Based on this information we build a propagation graph where each node identifies a platform and a transition between two nodes identifies a propagation between the nodes. A probability is associated to each transition to characterize its likelihood of occurrence.

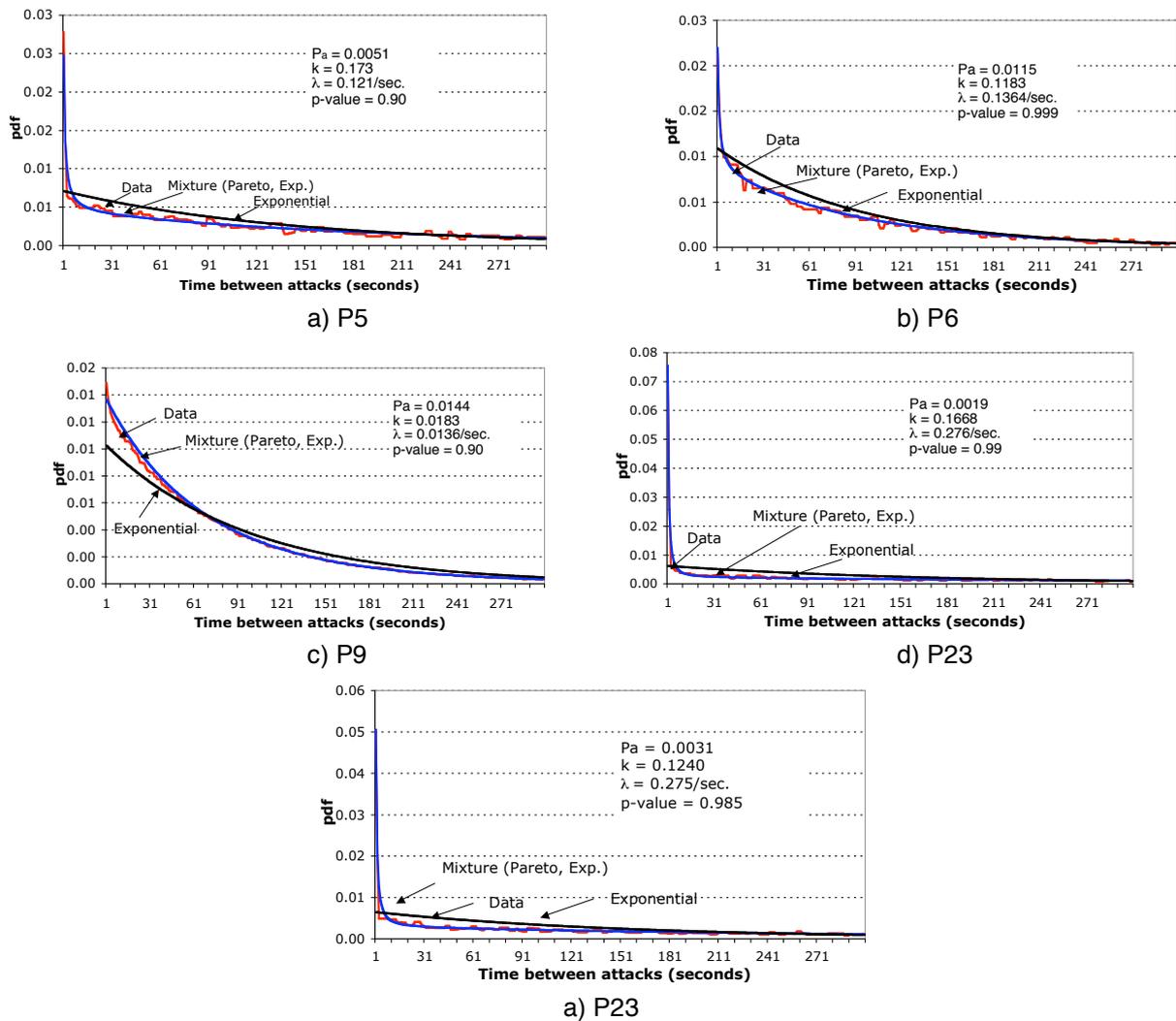


Figure 3- Observed and estimated times between attacks probability density functions.

Figure 4 presents the propagation graph obtained for the five platforms included in the analysis. Considering platforms P6 and P20, it can be seen that only a few IP addresses that attacked these platforms have been observed on the other platforms. The situation is different when considering platforms P5, P9, and P23. In particular, it can be noticed that propagation between P5 and P9 is highly probable. This is related in particular to the fact that the addresses of the corresponding platforms belong to the same /8 network domain. More thorough and detailed analyses are currently carried out based on the propagation graph in order to take into account timing information for the corresponding transitions and also the types of attacks observed, in order to better explain the propagation phenomena illustrated by the graph.

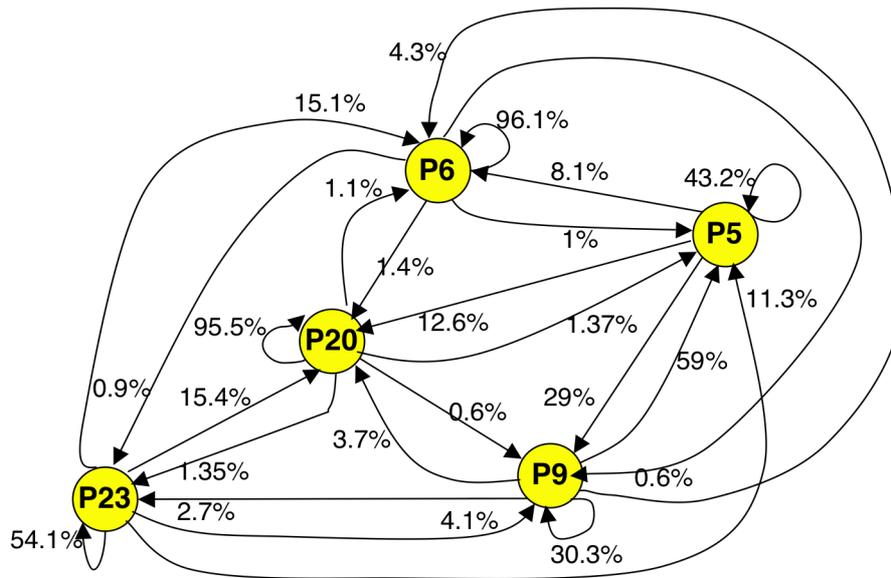


Figure 4- Propagation graph

Conclusion

This paper presented simple examples and preliminary models illustrating various types of empirical analysis and modeling activities that can be carried out based on the data collected from honeypots in order to characterize attack processes. The honeypot platforms deployed so far in our project belong to the family of so-called “low interaction honeypots”. Thus, hackers can only scan ports and send requests to fake servers without ever succeeding in taking control over them. In our project, we are also interested in running experiments with “high interaction” honeypots where attackers can really compromise the targets. Such honeypots are suitable to collect data that would enable us to study the behaviors of attackers once they have managed to get access to a target and try to progress in the intrusion process to get additional privileges. Future work will be focused on the deployment of such honeypots and the exploitation of the collected data to better characterize attack scenarios and analyze their impact on the security of the target systems. The ultimate objective would be to build representative stochastic models that will enable us to evaluate the ability of computing systems to resist to attacks and to validate them based on real attack data.

Acknowledgement. This work was partially supported by: 1) CADHo, a research action funded by the French ACI “Sécurité & Informatique” (www.cadho.org), and 2) the ReSIST European Network of Excellence IST-026764 (www.resist-noe.org).

References

- [1] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, "The Internet Motion Sensor: A Distributed Blackhole Monitoring System," Proc. 12th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, Feb. 2005.
- [2] Home Page of the CAIDA Project, <http://www.caida.org/>
- [3] DShield Distributed Detection System homepage, <http://www.honeynet.org/>
- [4] E. Alata, M. Dacier, Y. Deswarte, M. Kaâniche, K. Kortchinsky, V. Nicomette, V.H. Pham, F. Pouget, "Collection and Analysis of Attack data based on honeypots deployed on the Internet", 1st Workshop on Quality of Protection, Milano, Italy, September 2005.
- [5] F. Pouget, M. Dacier, V. H. Pham, "Leurré.com: On the Advantages of Deploying a Large Scale Distributed Honeypot Platform", Proc. E-Crime and Computer Evidence Conference (ECCE 2005), Monaco, Mars 2005.
- [6] L. Spitzner, Honeypots: Tracking Hackers, Addison-Wesley, ISBN from-321-10895-7, 2002
- [7] Project Leurré.com. Publications web page, <http://www.leurrecom.org/paper.htm>
- [8] M. Dacier, F. Pouget, H. Debar, "Honeypots: Practical Means to Validate Malicious Fault Assumptions on the Internet", Proc. 10th IEEE International Symposium Pacific Rim Dependable Computing (PRDC10), Tahiti, March 2004, pages 383-388.
- [9] M. Dacier, F. Pouget, H. Debar, "Attack Processes found on the Internet", Proc. OTAN Symp. on Adaptive Defense in Unclassified Networks, Toulouse, France, April 2004.
- [10] Honeyd Home page, <http://www.citi.umich.edu/u/provos/honeyd/>
- [11] R statistical package Home page, <http://www.r-project.org>

Dependability Benchmarks for Operating Systems*

Karama Kanoun and Yves Crouzet

LAAS-CNRS

Abstract

Dependability evaluation is playing an increasing role in system and software engineering together with performance evaluation. Performance benchmarks are widely used to evaluate system performance while dependability benchmarks are hardly emerging. A dependability benchmark for operating systems is intended to objectively characterize the operating system's behavior in the presence of faults, through dependability and performance-related measures, obtained by means of controlled experiments. This paper presents a dependability benchmark for general-purpose operating systems and its application to three versions of Windows operating system and four versions of Linux operating system. The benchmark measures are: operating system robustness (as regards possible erroneous inputs provided by the application software to the operating system via the application programming interface), operating system reaction and restart times in the presence of faults. The workload is JVM (Java Virtual Machine), a software layer, on top of the operating system allowing applications in Java language to be platform independent.

1. Introduction

Software dependability is usually evaluated based on data related to failures and corrections, observed on the software under development or during its operational life. However, when considering Off-The-Shelf software systems (which is the case of operating systems, OSs), most of the time no dependability data is available from their development. Only data collected during operation (if available) can be used to evaluate their dependability, which may be too late for selecting the right OS for building a new computer system based on an OS. In which case controlled experiments are of great help. The latter can either be carried out case-by-case (i.e., *ad hoc* way) or in a well-structured and standardized way, in order to characterize objectively the system behavior in the presence of faults. This is the aim of dependability benchmarks. Benchmarking the dependability of a system consists in evaluating dependability or performance-related measures, experimentally or based on experimentation and modeling, in a standard way. To be meaningful, a benchmark must satisfy a set of properties (e.g., representativeness, reproducibility, repeatability, portability, cost effectiveness). These properties must be taken into consideration from the earliest phases of the benchmark specification.

Our dependability benchmark is a robustness benchmark. Robustness is defined as the degree to which a system operates correctly in the presence of exceptional inputs or stressful environmental conditions. Robustness of OS can be viewed as its capacity to resist/react to faults induced by the applications running on top of it, or originating from the hardware layer or from device drivers.

* Has been published in International Journal of Performability Engineering, Vol. 2, No. 3, July 2006, 275-287

We address here the OS robustness as regards possible erroneous inputs provided by the application software to the OS via the Application Programming Interface (API). More explicitly, we consider corrupted parameters in system calls, shortly referred to as *faults*.

The benchmark presented in this paper is based on JVM (Java Virtual Machine), a software layer on top of the OS, allowing applications in Java language to be platform independent. It is applied to three Windows and four Linux OSs. The main concepts of the benchmark have been developed within the European project on Dependability Benchmarking, DBench [1].

The set of JVM dependability benchmarks is to the third set of OS benchmarks we have built up for Windows and Linux, based on the same high-level specification of the benchmark, using different workloads. The two previous ones used TPC-C Client performance benchmark for transactional systems [2] and PostMark, a file system performance benchmark [3]. Sensitivity analyses of the results with respect to the OS family benchmarked, the workload used and the faultload applied helped us to gain progressively confidence in the benchmark specification and in the results obtained.

The work reported in [4] is the most similar to ours, it addressed the "non-robustness" of the POSIX and Win32 APIs. Pioneer work on robustness benchmarking is published in [5]. Since then, a few studies have addressed OS dependability benchmarks, considering real time microkernels [6-8] or general purpose OSs [9, 10]. Robustness with respect to faults in device drivers is addressed in [11-13].

The remainder of the paper is organized as follows. Section 2 gives the specification of our OS benchmark. Section 3 is devoted to the benchmark implementation for Windows and Linux families. Section 4 presents benchmark results. Section 5 addresses benchmark properties and Section 6 concludes the paper.

2. Specification of the Benchmark

A dependability benchmark is specified through the definition of i) the benchmark target, ii) measures to be evaluated, iii) benchmark execution profile to be used to activate the operating system, iv) guidelines for conducting benchmark experiments and implementing the benchmark. The benchmark results are meaningful, useful and interpretable only if all the above items are supplied together with the results.

2.1. Benchmarking Target

An OS is a generic software layer providing basic services to the applications through the API, and communication with peripherals devices via device drivers. The *benchmark target* corresponds to the OS with the minimum set of device drivers necessary to run the OS under the benchmark execution profile. However, the benchmark target runs on a hardware platform whose characteristics impact the results. Thus, all benchmarks must be performed on the same hardware platform.

Although, in practice, the benchmark measures characterize the target system and the hardware platform, we state simply that the benchmark results characterize the OS.

Our benchmark addresses the user perspective, i.e., it is intended to be performed by (and to be useful for) someone who has no thorough knowledge about the OS and whose aim is to improve her/his knowledge about its behavior in the presence of faults. In practice, the user may well be the developer or the integrator of a system including the OS.

As a consequence, the OS is considered as a “black box”. The only required information is its description in terms of system calls and in terms of services provided.

2.2. Benchmark Measures

The OS receives a corrupted system call. After execution of such a call, the OS is in one of the following states:

S_{Er} (Error code): the OS generates an error code that is delivered to the application.

S_{Xp} (Exception): in the user mode, the OS processes the exception and notifies the application. However, for some critical situations, the OS aborts the application. In the kernel mode an exception is automatically followed by a *panic* state (e.g., blue screen for Windows and oops messages for Linux). Hence, the latter exceptions are included in the panic state and the term exception refers only to the first case of user mode exception.

S_{Pc} (panic): the OS is still “alive” but it is not servicing the application. In some cases, a soft reboot is sufficient to restart the system.

S_{Hg} (Hang): a hard reboot of the OS is required.

S_{NS} (No Signaling): the OS does not detect the erroneous parameter and executes the erroneous system call. S_{NS} is presumed when none of the previous situations (S_{Er}, S_{Xp}, S_{Pc}, S_{Hg}) is observed.

Panic and *hang* situations (S_{Pc}, S_{Hg}) are actual states in which the OS can stay for a while. S_{Er} and S_{Xp} characterize events. They are easily identified when the OS provides an error code or notifies an exception.

The benchmark measures include a robustness measure and two temporal measures.

OS Robustness (POS) is defined as the percentages of experiments leading to any of the states listed above. POS is thus a vector composed of 5 elements.

Reaction Time (T_{reac}) corresponds to the average time necessary for the OS to respond to a system call in presence of faults, either by notifying an exception or by returning an error code or by executing the required instructions.

Restart Time (T_{res}) corresponds to the average time necessary for the OS to restart after the execution of the workload in the presence of faults. Although under nominal operation the OS restart time is almost deterministic, it may be impacted by the corrupted system call. The OS might need additional time to make the necessary checks and recovery actions, depending on the impact of the fault applied.

The OS *reaction time* and *restart time* are also evaluated by experimentation in absence of faults for comparison purposes. They are respectively denoted *treac* and *tres*.

2.3. Benchmark Execution Profile

For performance benchmarks, the benchmark execution profile is a workload that is as realistic and representative as possible for the system under benchmarking. For a dependability benchmark, the execution profile includes, in addition to the workload, a set of faults, referred to as the faultload.

In the current benchmark, the workload is JVM, solicited through a program allowing to display «*Hello World*» on the screen. This program activates 76 system calls for Windows family and 31 to 37 system calls for Linux Family.

The faultload consists of corrupted parameters of system calls. For Windows, system calls are provided to the OS through the Win32 environment subsystem. In Linux OSs, these system calls are provided to the OS via the POSIX API. During runtime, the system calls activated by the workload are intercepted, corrupted and re-inserted.

The parameter corruption technique relies on thorough analyses of system call parameters to define *selective substitutions* to be applied to these parameters (similarly to the one used in [14]). A parameter is either a data or an address. The value of a data can be substituted either by an *out-of-range* value or by an *incorrect* (but not out-of-range) value, while an address is substituted by an *incorrect* (but existing) address (that could contain an incorrect or out-of-range data). We use a mix of these three techniques. More details can be found in [3].

2.4. Benchmark Conduct

Since disturbing the operating system may lead the OS to hang, a remote machine, referred to as the benchmark controller, is required to control the benchmark experiments, mainly in case of OS Hang or Panic states or workload hang or abort states (that cannot be reported by the machine hosting the benchmark target). Hence, we need at least two computers as shown in Figure 1. The *Target Machine* hosts the benchmarked OS and the workload, and ii) the *Benchmark Controller* is in charge of diagnosing and collecting part of benchmark data.

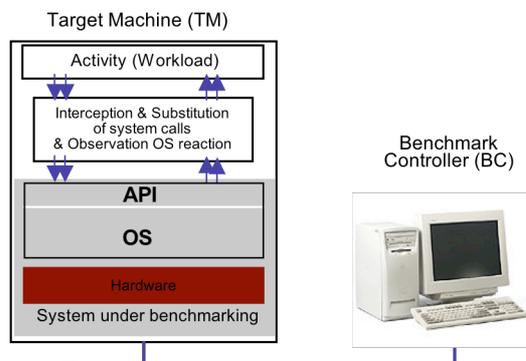


Figure 1: Benchmark Environment

The two machines perform the following: i) restart of the system before each experiment and launch of the workload, ii) interception of system calls with parameters, iii) corruption of system call parameters, iv) re-insertion of corrupted system calls, v) observation and collection of OS states. The experiment steps in case of workload completion are illustrated in Figure 2 and will be detailed in the next section. In case of workload non-completion state (i.e., the workload is in abort or hang state), the end of the experiment is governed by a watchdog timeout, fixed to 3 times the workload execution time without faults.

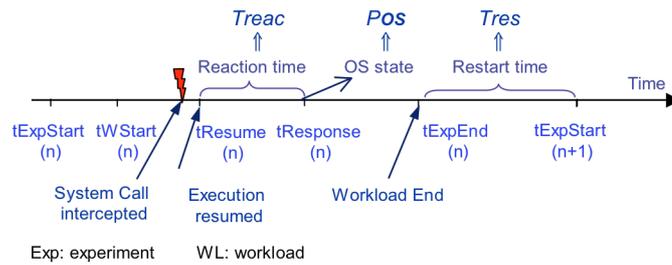


Figure 2: Benchmark Execution Sequence in Case of Workload Completion

3. Benchmark Implementation

3.1. Prototype

In order to obtain comparable results, all the experiments are run on the same target machine, composed of an Intel Pentium III Processor, 800 MHz, and a memory of 512 Mega Bytes. The hard disk is 18 Giga Bytes, ULTRA 160 SCSI. The benchmark controller in both prototypes for Windows and Linux is a Sun Microsystems workstation.

To intercept Win32 functions, we use the Detours tool [15], a library for intercepting arbitrary Win32 binary functions on X86 machines. We added three modules for i) substituting parameters of system calls by corrupted values ii) observing the reactions of the OS after execution of a corrupted system call, and iii) collecting the required measurements.

To intercept POSIX system calls, we used another interception tool, Strace [16] to which we added modules similar to those added to Detours.

3.2. Benchmark Preparation

Before each benchmark run (i. e., execution of the series of experiments related to a given OS), the target kernel is installed, and the interceptor is compiled for the current kernel (interceptors are kernel-dependent both for Windows and Linux). Once the benchmarking tool is compiled, it is used to identify the set of system calls activated by the workload. Parameters of these system calls are then analyzed and a database of corrupted values is built accordingly.

3.3. Benchmark Execution

At the beginning of each experiment, the target machine (TM) records the experiment start instant $t_{ExpStart}$ and sends it to the benchmark controller (BC) along with a notification of experiment start-up. The workload starts its execution. The Observer module records, in the experiment execution trace, the start-up instant of the workload, t_{WStart} , the activated system calls and their responses. This trace also collects the relevant data concerning states SEr , SXp and SNS . The recorded trace is sent to the BC at the beginning of the next experiment.

The parameter substitution module identifies the system call to be corrupted. The execution is then interrupted, a parameter value is substituted and the execution is resumed with the corrupted parameter value

(t_{Resume} is saved in the experiment execution trace). The state of the OS is monitored so as to diagnose SEr, SXp, SNS. The corresponding OS response time ($t_{Response}$) is recorded in the experiment execution trace. For each run, the OS reaction time after the experiment is calculated as the difference between $t_{Response}$ and t_{Resume} . At the end of the execution of the workload, the OS notifies the end of the experiment to the BC by sending an end signal along with the experiment end instant, t_{ExpEnd} . If the workload does not complete, then t_{ExpEnd} is governed by the value of a watchdog timer. If, at the end of the watchdog timer, the BC has not received the end signal from the OS, it then attempts to connect to the OS. If this connection is successful, and if the soft reboot is successful, then a workload abort or hang state is diagnosed. If the soft reboot is unsuccessful, then a panic state, SPc, is deduced and a hard reboot is required. Otherwise SHg is assumed.

At the end of a benchmark execution, all files containing raw results corresponding to all experiments are on the BC. A processing module extracts automatically the relevant information from these files (two specific modules are required for Windows and Linux families). The relevant information is then used to evaluate automatically the benchmark measures (the same module is used for Windows and Linux).

3.4. Benchmark Characteristics

For each system call activated by the workload, several parameters are corrupted leading to several experiments for the same system call. The number of system calls (activated by JVM under the program allowing to display «Hello World» on the screen) and the associated number of experiments for the OSs considered are indicated in Table 1.

Table 1: Number of System Calls and Experiments for each OS

	Windows family			Linux family			
	W- NT4	W- 2000	W- XP	L- 2.2.26	L- 2.4.5	L- 2.4.26	L- 2.6.6
# System Calls	76	76	76	37	32	32	31
# Experiments	1285	1294	1282	457	408	408	409

4. Benchmark Results

4.1. Measures

OSs robustness is given in Figure 3. It shows that all OSs of the same family are equivalent. It also shows that none of the catastrophic states (*Panic* or *Hang* OS states) occurred for all Windows and Linux OSs. Linux OSs notified more error codes (58-66%) than Windows (25%), while more exceptions were raised with Windows (22-23%) than with Linux (7-10%). More no-signaling cases have been observed for Windows (52-54%) than for Linux (27-36%).

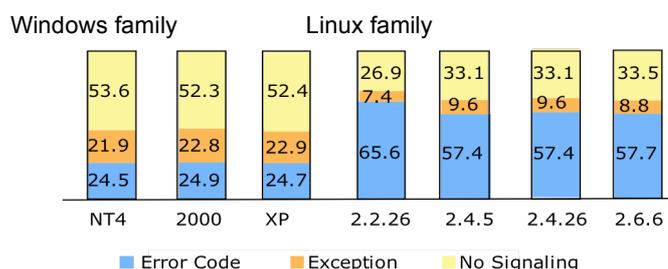


Figure 3: OS Robustness (%)

These results are in conformance with our previous results, related to Windows using TPC-C Client [2] and to Windows and Linux using PostMark [3]. In [4] it was observed that on the one hand Windows 95, 98, 98SE and CE had a few catastrophic failures and on the other hand Windows NT, Windows 2000 and Linux are more robust and did not have any catastrophic failures as in our case.

The reaction times in the presence of faults (and without fault) are given in Figure 4. Note that for the Windows family, XP has the lowest reaction time, and for the Linux family, 2.6.6 has the lowest one. However, the reaction times of Windows NT and 2000 are very high. A detailed analysis showed that the large response time for Windows NT and 2000 are mainly due to system calls LoadLibraryA, LoadLibraryExA and LoadLibraryEXW. Not including these system calls when evaluating the average of the reaction time in the presence of faults leads respectively to 388 μ s, 182 μ s and 205 μ s for NT4, 2000 and XP (the associated average restart times without fault become respectively 191 μ s, 278 μ s and 298 μ s). For Linux the high values of the reaction times in presence of faults are also due to three system calls (execve, getdents64, nanosleep). Not including the reaction times associated to these system calls leads respectively to 88 μ s, 241 μ s, 227 μ s and 88 μ s for Linux 2.2.26, 2.4.5, 2.4.26 and 2.6.6.

The restart times are shown in Figure 5. The average restart time without faults, τ_{res} , is always lower than the benchmark restart time (with faults), τ_{res} , but the difference is not significant. The standard deviation is very large for all OSs. Linux 2.2.26 and Windows XP have the lowest restart time (71 seconds, in the absence of fault) while Windows NT and 2000 restart times are around 90 seconds and those of Linux versions 2.4.5, 2.4.26 and 2.6.6 are around 80 seconds.

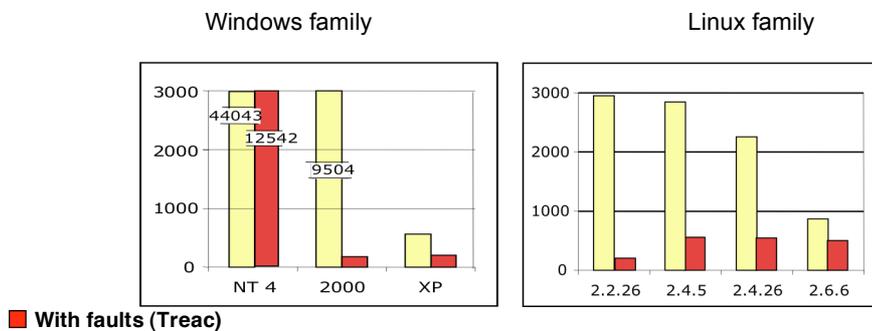


Figure 4: OS Reaction Times (in μ seconds)

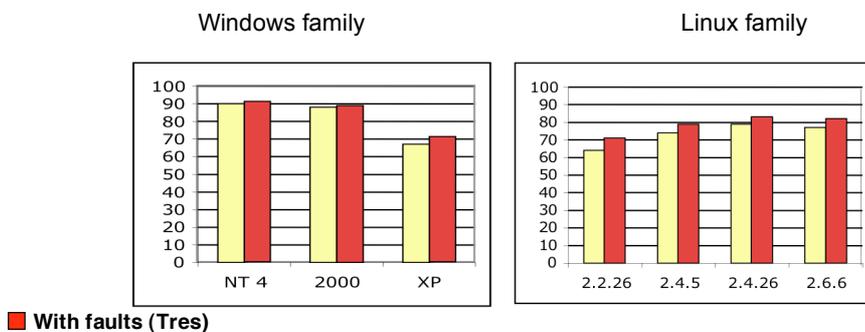


Figure 5: OS Restart Times (in seconds)

4.2. Restart Time Refinement

It is worth to mention that the average restart times mask interesting phenomena. Detailed analyses show that all OSs of the same family have similar behavior and that the two families exhibit very different behaviors.

For Windows, there is a correlation between the restart time and the workload state at the end of the experiment. When the workload is completed, the restart time is almost the same as the average restart time without substitution. On the other hand, the restart time is statistically larger for all experiments with workload abort/hang. Moreover, statistically, the same system calls lead to workload abort/hang.

This is illustrated in Figure 6 in which the benchmark experiments are executed in the same order for the three Windows versions. Similar behaviors have been observed when using TPC-C [2] and PostMark workloads [3].

Linux restart time is not affected by the workload state. Detailed restart time analyses show high values appearing periodically. These values correspond to a check-disk performed by the Linux kernel every 26 restarts (which explains the important standard deviation on this measure). This is illustrated in Figure 7 for Linux 2.2.26, as an example. The same behavior has been observed when using the PostMark workload [3].

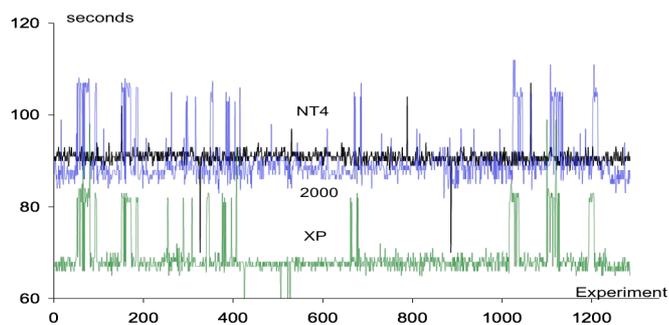


Figure 6: Detailed Restart Time for Windows

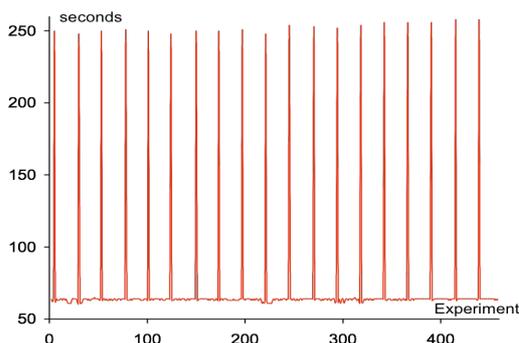


Figure 7: Detailed Restart Time for Linux 2.2.26

5. Benchmark Properties

To be accepted and adopted by the scientific and industrial communities, a benchmark must satisfy a set of key properties, such as representativeness, repeatability, reproducibility, portability and cost-effectiveness. Representativeness concerns essentially the workload (that is without any doubt the most critical component of any dependability benchmark). All properties should be accounted for from the early phase of the benchmark specification as they directly impact the specification of all benchmark components. Some properties can be ensured by construction, some others have to be checked experimentally too.

In our previous work using TPC-C and PostMark as workloads, we have shown how the various properties have been taken into consideration what has been done to check some of them experimentally. In this section, we summarize the various properties and the kinds of verification carried out using JVM.

5.1 Faultload representativeness

It is very hard to guarantee that the faults used in our benchmark (corrupted values in system call parameters) are representative of all application software faults. Indeed, the OS is not expected to detect all application faults, but it is expected to avoid some application faults that may lead to OS misbehavior. At least it is expected to detect system calls with obvious errors (such as out-of-range data or incorrect addresses). We have thus performed sensitivity analyses with respect to the parameter corruption technique.

The selective substitution technique used is composed of a mix of three corruption techniques as mentioned in Section 2.3: out-of-range data (OORD), incorrect data (ID) and incorrect addresses (IA). Let us denote the faultload used in our benchmarks by FL0. To analyze the impact of the faultload, we consider two subsets, including respectively i) IA and OORD only (denoted FL1), and ii) OORD only (denoted FL2). Taking Windows NT4 and Linux 2.2.26 as examples, moving from FL0 to FL2 the number of experiments decreases respectively from 1285 to 264 and from 457 to 119.

We ran the benchmarks of all OSs considered using successively FL0, FL1 and FL2. The results obtained confirm the equivalence between Linux family OSs as well as the equivalence between Windows family OSs, using the same faultload (FL0, FL1 or FL2). Indeed, for each OS, its robustness with respect to FL0, FL1 or FL2 is different but the robustness of all OSs of the same family with respect to each of the three faultloads is equivalent. The same results have been obtained using TPC-C Client and PostMark as workloads. This shows that using a mix of the three corruption techniques is meaningful.

5.2. Repeatability and Reproducibility

Repeatability is the property that guarantees *statistically equivalent results* when the benchmark is run more than once in the *same environment* (i.e., using the same system under benchmark and the same prototype). Our OS dependability benchmark is composed of a series of experiments. Each experiment is run after a system restart. The experiments are independent from each other and the order in which the experiments are run is not important at all. Hence, once the system calls to be corrupted are selected and the substitution values defined, the benchmark is fully repeatable. We have repeated all the benchmarks presented three times to check for repeatability.

Reproducibility is the property that guarantees that *another party* obtains statistically equivalent results when the benchmark is implemented from the *same specification* and is used to benchmark the same system

under benchmarking. Reproducibility is strongly related to the amount of details given in the specification. The specification should be at the same time i) *general enough* to be applied to the class of systems addressed by the benchmark and ii) *specific enough* to be implemented without distorting the original specification. We managed to satisfy such a tradeoff. Unfortunately, we have not checked explicitly the reproducibility of the benchmark results by developing several prototypes by different people. On the other hand, the results seem to be independent from the faultload. This makes us confident about reproducibility.

5.3. Portability

Portability concerns essentially the faultload (i.e., its applicability to different OS families).

At the specification level, in order to ensure portability of the faultload, the system calls to be corrupted are not identified individually. We decided to corrupt all system calls of the workload. This is because OSs from different families do not necessarily comprise the very same system calls as they may have different APIs. However, most OSs feature comparable functional components.

At the implementation level, portability can only be ensured for OSs from the same family because different OS families have different API sets.

5.4. Cost

If a benchmark is very expensive, industry may not be ready to adopt it. Cost is expressed in terms of effort required to develop the benchmark, run it and obtain results. These steps require some effort that is, from our point of view, relatively affordable. In our case, most of the effort was spent in defining the concepts, characterizing the faultload and studying its representativeness.

The JVM benchmark benefited a lot from TPC-C and PostMark benchmarks as all benchmark components did exist and we had only to adapt them. The first step consisted in executing JVM for each OS to be benchmarked, to identify system calls activated. The second step was devoted to define, for each system call, the parameters to be corrupted and the exact substitution values, to prepare the database to be used in the Interception /substitution/ observation modules. This step took a couple of days for Linux family (activating 31-37 system calls depending on the version considered) and the double for Windows as it activates 76 system calls. Adaptation of the benchmark controller and of the Interception/substitution/observation modules required about one day for each family.

The benchmark duration ranges from one day for each Linux OS to less than three days for each Windows OS. More precisely, the duration of an experiment with workload completion is less than 3 minutes (including the time to workload completion and the restart time), while it is less than 6 minutes without workload completion (including the watchdog timeout and the restart time). Thus, an experiment lasts less than 5 minutes for all OSs. The series of experiments of a benchmark is fully automated.

6. Conclusions

The dependability benchmark presented in this paper is the third benchmark we have developed for Windows and Linux, based on the same high-level specification of the benchmark but using different workloads. The results obtained are in conformance with those obtained with the two other workloads and increase our confidence in the benchmark specification and in the results obtained.

This benchmark and more generally the three benchmarks developed and applied show that all OSs of the same family are equivalent. They also show that none of the catastrophic states of the OS (*Panic* or *Hang*) occurred for any of the Windows and Linux OSs considered.

Linux OSs notified more error codes than Windows while more exceptions were raised with Windows than with Linux. More no-signaling cases have been observed for Windows than for Linux.

Concerning the OS reaction time measure, results show a great variation around the average due to a minority of system calls with large execution times that dodge the average. When these system calls are not considered, the reaction times of all the OSs of the same family become equivalent.

With respect to the restart time measure, Linux seems to be globally faster compared to Windows even though Windows XP and Linux 2.2.26 have the same restart times. Detailed analysis of the restart time showed i) a correlation between Windows restart time and the workload final state (in case of workload *hang* or *abort*, the restart time is higher than in case of workload completion) and ii) that Linux performs a “check disk” after each 26 restarts after which the restart time is four times higher than the average.

We paid a particular attention to representativeness of faultload, and to the properties of repeatability, reproducibility, portability and cost effectiveness of the benchmark.

Acknowledgement

We would like to thank Jean Arlat who contributed to the first OS benchmark based on TPC-C Client. We are indebted to Ali Kalakech, Ana-Elena Rugina and Philippe Rumeau for their valuable contributions. They have implemented the series benchmarks based on TPC-C, PostMark and JVM, allowing to validate progressively the benchmark concepts, results and properties.

References

- [1]. EC DBench project (IST-2000-25425), <http://www.laas.fr/DBench>, project final short report.
- [2]. Kalakech A., K. Kanoun, Y. Crouzet and A. Arlat, *Benchmarking the Dependability of Windows NT, 2000 and XP*, Int. Conf. on Dependable Systems and Networks, Florence, Italy, pp. 681-686, 2004.
- [3]. Kanoun K., Y. Crouzet, A. Kalakech, A. E. Rugina and P. Rumeau, *Benchmarking the Dependability of Windows and Linux using PostMark Workloads*, 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'2005), Chicago (USA), pp.11-20, 2005.
- [4]. Shelton C. et al., *Robustness Testing of the Microsoft Win32 API*, Int. Conf. on Dependable Systems and Networks, New York, pp. 261-270, 2000.
- [5]. Mukherjee A. and D. P. Siewiorek, *Measuring Software Dependability by Robustness Benchmarking*, IEEE Trans. of Software Engineering, Vol. 23 (6), pp. 366-378, 1997.
- [6]. Chevochot P. and I. Puaut, *Experimental Evaluation of the Fail-Silent Behavior of a Distributed Real-Time Run-Time Support Built from COTS Components*, Proc. Int. Conference on Dependable Systems and Networks (DSN-2001), Göteborg, Sweden, 2001.
- [7]. Arlat J. et al., *Dependability of COTS Microkernel-Based Systems*, IEEE Trans. on Computers, Vol.51 (2), pp. 138-163, 2002.
- [8]. Gu W, Z. Kalbarczyk and R. K. Iyer, *Error Sensitivity of the Linux Kernel Executing on PowerPC G4 and Pentium 4 Processors*, Int. Conf. on Dependable Systems and Networks, Florence, Italy, pp. 887-896, 2004.
- [9]. Tsai T. K. et al., *An Approach Towards Benchmarking of Fault-Tolerant Commercial Systems*, 26th Int. Symp. on Fault-Tolerant Computing, Sendai, Japan, pp. 314-323, 1996.
- [10]. Koopman P. and J. DeVale, *Comparing the Robustness of POSIX Operating Systems*, 29th Int. Symp. on Fault-Tolerant Computing, Madison, pp. 30-37, 1999.
- [11]. Durães J. and H. Madeira, *Characterization of Operating Systems Behavior in the Presence of Faulty Drivers through Software Fault Emulation*, Pacific Rim Int. Sym. on Dependable Computing, Tsukuba, Japan, pp. 201-209, 2002.
- [12]. Chou A. et al., *An Empirical Study of Operating Systems Errors*, 18th ACM Symp. on Operating Systems Principles Banff, AL, Canada, pp. 73-88, 2001.

- [13]. Albinet A., J. Arlat and J.-C. Fabre, *Characterization of the Impact of Faulty Drivers on the Robustness of the Linux Kernel*, Int. Conf. on Dependable Systems and Network, Florence, Italy, pp. 867-876, 2004.
- [14]. Koopman P. et al., *Comparing Operating Systems using Robustness Benchmarks*, 16th Int. Symp. on Reliable Distributed Systems, Durham, USA, pp. 72-79, 1997.
- [15]. Hunt G. and D. Brubaker, *Detours: Binary Interception of Win32 Functions*, 3rd USENIX Windows NT Symp., Seattle, Washington, USA, pp. 135-144, 1999.
- [16]. McGrath R. and W. Akkerman, *Source Forge Strace Project*, <http://sourceforge.net/projects/strace>, 2004.

Windows and Linux Robustness Benchmarks With respect to Application Erroneous Behavior

Karama Kanoun, Yves Crouzet, Ali Kalakech and Ana-Elena Rugina
LAAS-CNRS, 7, Avenue Colonel Roche 31077 Toulouse Cedex 4, France

Abstract

This chapter presents the specification of dependability benchmarks for general-purpose operating systems with respect to application erroneous behavior, and shows examples of benchmark results obtained for various versions of Windows and Linux operating systems. The benchmark measures are: operating system robustness (as regards possible erroneous inputs provided by the application software to the operating system (OS) via the application programming interface), the OS reaction and restart times in the presence of faults. Two workloads are used for implementing the benchmark: PostMark, a file system performance benchmark for operating systems, and the Java Virtual Machine (JVM) middleware, a software layer on top of the OS allowing applications in Java language to be platform independent.

1. Introduction

Software is playing an increasingly important role in our day-to-day life. In particular, operating systems (OSs) are more and more used even in critical application domains. Choosing the operating system that is best adapted to one's needs is becoming a necessity. For a long time, performance was the main selection criterion for most users and several performance benchmarks were developed and are widely used. However, an OS should not only have good performance but also a high dependability level. Dependability benchmarks emerged as a consequence. Their role is to provide useful information regarding the dependability of software systems [Tsai *et al.* 1996, Brown & Patterson 2000, Chevochot & Puaut 2001, Brown *et al.* 2002, Zhu *et al.* 2002]. This chapter is devoted to the specification, application and validation of two dependability benchmarks of OSs using two different workloads: PostMark, a file system performance benchmark, and JVM (Java Virtual Machine), a software layer on top of the OS allowing applications in Java language to be platform independent.

Benchmarking the dependability of a system consists of evaluating dependability or performance-related measures, experimentally or based on experimentation and modeling, in order to characterize objectively the system behavior in the presence of faults. Such an evaluation should allow non-ambiguous comparison of alternative solutions. Non-ambiguity, confidence in results and meaningfulness are ensured by a set of properties a benchmark should satisfy. For example, a benchmark must be representative, reproducible, repeatable, portable and cost effective. These properties should be taken into consideration from the earliest phases of the benchmark specification as they have a deep impact on almost all benchmark components. Verification of the benchmark key properties constitutes a large part of the benchmark validation.

Our dependability benchmark is a robustness benchmark. Robustness is defined as the degree to which a system operates correctly in the presence of exceptional inputs or stressful environmental conditions. Robustness of OS can be viewed as its capacity to resist/react to faults induced by the applications running on top of it, or originating from the hardware layer or from device drivers. In this chapter we address the OS robustness as regards possible erroneous inputs provided by the application software to the OS via the Application Programming Interface (API). More explicitly, we consider corrupted parameters in system calls. For sake of conciseness, such erroneous inputs are referred to as *faults*.

The work reported in this chapter is a follow up of the European project on Dependability Benchmarking, DBench [DBench]. Our previously published work on OS dependability benchmarks was based on i) TPC-C Client performance benchmark for transactional systems [Kalakech *et al.* 2004b], ii) PostMark [Kanoun *et al.* 2005] and on JVM workload [Kanoun & Crouzet 2006].

The work reported in [Shelton *et al.* 2000] is the most similar to ours, it addressed the "non-robustness" of the POSIX and Win32 APIs (while we are interested in robust and non-robust behavior). Pioneer work on robustness benchmarking is published in [Mukherjee & Siewiorek 1997]. Since then, a few studies have addressed OS dependability benchmarks, considering real time microkernels [Chevochot & Puaut 2001, Arlat *et al.* 2002, Gu *et al.* 2004] or general purpose OSs [Tsai *et al.* 1996, Koopman & DeVale 1999]. Robustness with respect to faults in device drivers is addressed in [Chou *et al.* 2001, Durães & Madeira 2002, Albinet *et al.* 2004].

The remainder of the chapter is organized as follows. Section 2 gives the specification of the OS benchmarks. Section 3 presents benchmark implementation and results related to PostMark workload for Windows and Linux families. Section 4 presents benchmark implementation and results related to JVM workload. Section 5 refines the benchmark results for PostMark and JVM. Section 6 outlines the main

benchmark properties that are meaningful to OS benchmarks, and briefly shows what has been achieved to ensure and check them. Section 7 concludes the chapter.

2. Specification of the Benchmark

In order to provide dependability benchmark results are meaningful, useful and interpretable it is essential to define clearly the following benchmark components:

- 1) The benchmarking context.
- 2) The benchmark measures to be evaluated and the measurements to be performed on the system to provide the information required for obtaining them.
- 3) The benchmark execution profile to be used to activate the operating system.
- 4) Guidelines for conducting benchmark experiments and implementing benchmark prototypes.

These components are presented hereafter..

2.1. Benchmarking Context

An OS can be seen as a generic software layer that manages all aspects of the underlying hardware. The OS provides i) basic services to the applications through the API, and ii) communication with peripheral devices via device drivers. From the viewpoint of dependability benchmarking, the *benchmark target* corresponds to the OS with the minimum set of device drivers necessary to run the OS under the benchmark execution profile. However, for the benchmark target to be assessed, it is necessary to run it on top of a hardware platform and to use a set of libraries. Thus, the benchmark target along with the hardware platform and libraries form the *system under benchmarking*. Although, in practice, the benchmark measures characterize the system under benchmarking (e.g., the OS reaction and restart times are strongly dependent on the underlying hardware), for clarity purpose we will state that the benchmark results characterize the OS.

The benchmark addresses the user perspective, i.e., it is primarily intended to be performed by (and to be useful for) someone or an entity who has no in depth knowledge about the OS and whose aim is to significantly improve her/his knowledge about its behavior in the presence of faults. In practice, the user may well be the developer or the integrator of a system including the OS.

The OS is considered as a “black box” and the source code does not need to be available. The only required information is the description of the OS in terms of system calls (in addition of course to the description of the services provided by the OS).

2.2. Benchmark Measures

The benchmark measures include a robustness measure and two temporal measures.

After execution of a corrupted system call, the OS is in one of the states summarized in Table 1.

SEr	An <i>error code</i> is returned
SXp	An <i>exception</i> is raised, processed and notified
SPc	<i>Panic</i> state
SHg	<i>Hang</i> state
SNS	No-signaling state

Table 1: OS outcomes

SEr: corresponds to the case where the OS generates an error code that is delivered to the application.

SXp: corresponds to the case where the OS issues an exception. Two kinds of exceptions can be distinguished depending on whether it is issued during the application software execution (user mode) or during execution of the kernel software (kernel mode). In the user mode, the OS processes the exception and notifies the application (the application may or may not take into account explicitly this information). However, for some critical situations, the OS aborts the application. An exception in the kernel mode is automatically followed by a *panic* state (e.g., blue screen for Windows and oops messages for Linux). Hence, hereafter, the latter exceptions are included in the panic state and the term exception refers only to user mode exceptions.

SPc: In the panic state, the OS is still “alive” but it is not servicing the application. In some cases, a soft reboot is sufficient to restart the system.

SHg: In this state, a hard reboot of the OS is required.

SNS: In the no-signaling state, the OS does not detect the presence of the erroneous parameter. As a consequence, it accepts the erroneous system call and executes it. It may thus abort, hang or complete its execution. However, the response might be erroneous or correct. For some system calls, the application may not require any explicit response, so it simply continues execution after sending the system call. SNS is presumed when none of the previous outcomes (SEr, SXp, SPc, SHg) is observed.

Panic and *hang* outcomes are actual states in which the OS can stay for a while. They characterize the OS’s non-robustness. Conversely, SEr and SXp characterize only events. They are easily identified when the OS provides an error code or notifies an exception. These events characterize the OS’s robustness.

OS Robustness (POS) is defined as the percentages of experiments leading to any of the outcomes listed in Table 1. POS is thus a vector composed of 5 elements.

Reaction Time (*T_{reac}*) corresponds to the average time necessary for the OS to respond to a system call in the presence of faults, either by notifying an exception or by returning an error code or by executing the system call.

Restart Time (*T_{res}*) corresponds to the average time necessary for the OS to restart after the execution of the workload in the presence of one fault in one of its system calls. Although under nominal operation the OS restart time is almost deterministic, it may be impacted by the corrupted system call. The OS might need additional time to make the necessary checks and recovery actions, depending on the impact of the fault applied.

The OS *reaction time* and *restart time* are also observed in absence of faults for comparison purpose. They are respectively denoted τ_{reac} and τ_{res} .

Note that our set of measures is different of the one used in [Shelton et al. 2000]. Shelton et al. only take into account non-robust behavior of the OS. They use the CRASH scale to measure the OS non-robustness (Catastrophic, **R**estart, **A**bort, **S**ilent and **H**indering failures). Our aim is to distinguish as clearly as possible

the proportions of robust versus non-robust behavior of the OS. Also, we completed the set of measures by adding to it the two temporal measures (*Texec* and *Tres*) presented above.

2.3. Benchmark Execution Profile

For performance benchmarks, the benchmark execution profile is a workload that is as realistic and representative as possible for the system under benchmarking. For a dependability benchmark, the execution profile includes, in addition, corrupted parameters in system calls. The set of corrupted parameters is referred to as the faultload.

The benchmark is defined so that the workload could be any performance benchmark workload (and, more generally, any user specific application) intended to run on top of the target OS. In [Kalakech *et al.* 2004b] we have used the workload of TPC-C Client [TPC-C 2002], and in this work we use two workloads PostMark [Katcher 1997] and JVM [Lindholm & Yellin 1999]. The two workloads will be discussed further in Sections 3 and 4.

The faultload consists of corrupted parameters of system calls. For Windows, system calls are provided to the OS through the Win32 environment subsystem. For Linux OSs, these system calls are provided to the OS via the POSIX API. During runtime, the workload system calls are intercepted, corrupted and re-inserted.

We use a parameter corruption technique relying on thorough analysis of system call parameters to define *selective substitutions* to be applied to these parameters (similar to the one used in [Koopman *et al.* 1997]). A parameter is either a data or an address. The value of a data can be substituted either by an *out-of-range* value or by an *incorrect* (but not out-of-range) value, while an address is substituted by an *incorrect* (but existing) address (containing usually an incorrect or out-of-range data). We use a mix of these three corruption techniques. Note that non-existing addresses are always detected. Hence they are not considered as interesting substitution values.

To reduce the number of experiments, the parameter data types are grouped into classes. A set of substitution values is defined for each class. They depend on the definition of the class. Some values require a *pre* and a *post* processing such as the creation and the destruction of temporary files. For example, for Windows, we group the data types into 13 classes. Among these classes, 9 are pointer classes. Apart from *pvoid* (pointer which points to anything), all other pointers point to a particular data type. Substitution values for these pointers are combination of pointer substitution values and the corresponding data type substitution values. Similarly, for Linux, we group the data types into 13 classes among which 5 are pointer classes. We use the same substitution values for basic data types (i.e., integer) both for Windows and Linux. Nevertheless, some data types are system-dependent. Consequently, they have specific substitution values. In Linux, for example, we define a class corresponding to the type `mode`. A mode is an integer with a particular meaning: read/write modes or permission flags. As the validity domain of this data type can be identified precisely, pertinent substitution values are defined for it. Table 2 reviews the substitution values associated with the basic data type classes.

2.4. Benchmark Conduct and Implementation

Since perturbing the operating system may lead the OS to hang, a remote machine, referred to as the benchmark controller, is required to reliably control the benchmark experiments, mainly in case of OS Hang or Panic states or workload hang or abort states (that cannot be reported by the machine hosting the benchmark target). Accordingly, for running an OS dependability benchmark we need at least two computers: i) the *Target Machine* for hosting the benchmarked OS and the workload, and ii) the *Benchmark Controller* that is in charge of diagnosing and collecting part or all benchmark data. The two machines

perform the following functions: i) restart of the system before each experiment and launch of the workload, ii) interception of system calls with parameters, ii) corruption of system call parameters, iii) re-insertion of corrupted system calls, iv) observation and collection of OS outcomes.

Data type	class	Substitution values					
Pvoid		NULL	0xFFFFFFFF	1	0xFFFF	-1	Random
Integer		0	1	MAX INT	MIN INT	0.5	
Unsigned integer		0	1	0xFFFFFFFF	-1	0.5	
Boolean		0	0xFF (Max)	1	-1	0.5	
String		Empty	Large (> 200)	Far (+ 1000)			

Table 2: Parameter substitution values

The experiment steps in case of workload completion are illustrated in Figure 1. In case of workload non-completion state (i.e., the workload is in abort or hang state), the end of the experiment is provided by a watchdog timeout as illustrated in Figure 2. The timeout duration is fixed to a value that is three times greater than the largest workload execution time without faults.

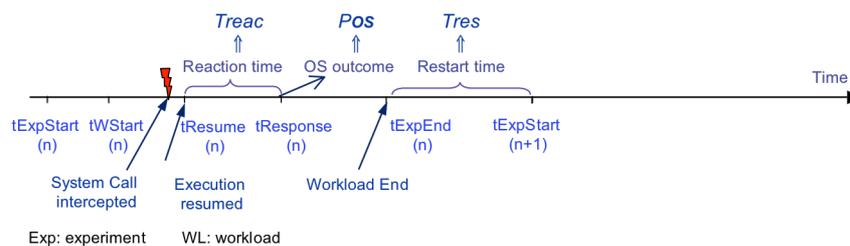


Figure 1: Benchmark execution sequence in case of workload completion

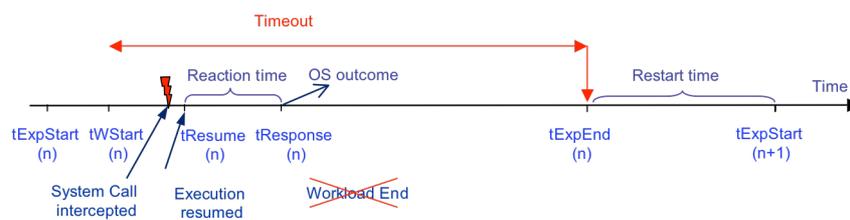


Figure 2: Benchmark execution sequence in case of workload abort or hang

To intercept Win32 functions, we use the Detours tool [Hunt & Brubaher 1999], a library for intercepting arbitrary Win32 binary functions on X86 machines. The part of Detours in charge of system call interception is composed of 30 Kilo lines of code (KLOC). The modifications we carried out on this tool concern i) the replacement of system call parameters by corrupted values (this module is 3 KLOC) and ii) the addition of modules to observe the reactions of the OS after parameter corruption, and to collect the required measurements (this module is 15 KLOC). To intercept POSIX system calls, we used another interception tool, Strace [McGrath & Akkerman 2004]. Strace is composed of 26 KLOC. Also, we added two modules to this tool to allow i) substitution of the parameters and ii) observation of Linux behavior after parameter corruption (these modules correspond to 4 KLOC together). The reaction time is counted from the time the

corrupted system call is re-inserted. Hence the time to intercept and substitute system calls is not included in the system reaction time, as shown in Figures 1 and 2.

Figure 3 summarizes the various components of the benchmark environment. All the experiments have been run on the same target machine, composed of an Intel Pentium III Processor, 800 MHz, and a memory of 512 Mega Bytes. The hard disk is 18 Giga Bytes, ULTRA 160 SCSI. The benchmark controller in both prototypes for Windows and Linux is a Sun Microsystems workstation.

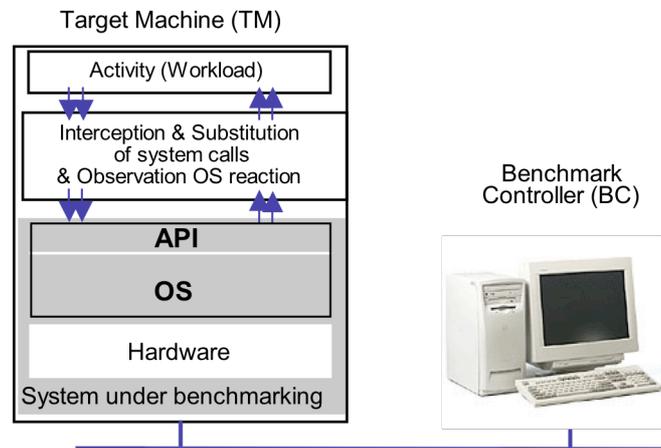


Figure 3. Benchmark environment

Before each benchmark run (i. e., before execution of the series of experiments related to a given OS), the target kernel is installed, and the interceptor is compiled for the current kernel (interceptors are kernel-dependent both for Windows and Linux as they depend on kernel headers that are different from one version to another). Once the benchmarking tool is compiled, it is used to identify the set of system calls activated by the workload. All parameters of all these system calls are then analyzed and placed into the corresponding class. A database of substitution values is then generated accordingly.

Following the benchmark execution sequence presented in Figures 1 and 2, at the beginning of each experiment, the target machine (TM) records the experiment start instant $t_{ExpStart}$ and sends it to the benchmark controller (BC) along with a notification of experiment start-up. The workload starts its execution. The Observer module records, in the experiment execution trace, the start-up instant of the workload, the activated system calls and their responses. This trace also collects the relevant data concerning states SEr , SXp and SNS . The recorded trace is sent to the BC at the beginning of the next experiment.

The parameter substitution module checks whether the current system call has parameters. If it is not the case, the execution is simply resumed; otherwise, the execution is interrupted, a parameter value is substituted and the execution is resumed with the corrupted parameter value (t_{Resume} is saved in the experiment execution trace). The state of the OS is monitored so as to diagnose SEr , SXp , SNS . The corresponding OS response time ($t_{Response}$) is recorded in the experiment execution trace. For each run, the OS reaction time is calculated as the difference between $t_{Response}$ and t_{Resume} .

At the end of the execution of the workload, the OS notifies the end of the experiment to the BC by sending an end signal along with the experiment end instant, t_{ExpEnd} and then it restarts so that the current experiment does not have any effects on the following experiment. If the workload does not complete, then t_{ExpEnd} is governed by the value of a watchdog timer. The BC collects the SHg state and the workload abort/hang states. It is in charge of restarting the system in such cases. When no faultload is applied, the average time necessary for the OS to execute PostMark or JVM is less than 1 minute for Windows and for

Linux. We considered that three times the normal execution time is enough to conclude on the experiment's result. Thus, we have fixed the watchdog timer to 3 minutes. If, at the end of this watchdog timer, the BC has not received the end signal from the OS, it then attempts to ping the OS. If the OS responds (ping successful), the BC attempts to connect to it. If the connection is successful, then a workload abort or hang state is diagnosed. If the connection is unsuccessful, then a panic state, SPc, is deduced. Otherwise, SHg is assumed. If workload abort / hang or SPc or SHg are observed, $t_{Response}$ does not take any value for the current experiment. Thus, the measure Treac is not skewed by the watchdog timer value.

At the end of a benchmark execution, all files containing raw results corresponding to all experiments are on the BC. A processing module extracts automatically the relevant information from these files (two specific modules are required for Windows and Linux families). The relevant information is then used to evaluate automatically the benchmark measures (the same module is used for Windows and Linux).

3. PostMark Dependability Benchmark Implementation and Results

PostMark creates a large pool of continually changing files and measures the transaction rates for a workload emulating Internet applications such as e-mail or netnews. It generates an initial pool of random text files ranging in size from a configurable low bound to a configurable high bound. The file pool is of configurable size and can be located on any accessible file system. The workload of this benchmark, referred to as PostMark for simplicity, is responsible for realizing a number of transactions. Each transaction consists of a pair of smaller transactions: i) create file or delete file and ii) read file or append file. PostMark is developed in C language.

From a practical point of view PostMark needs to be compiled separately for each OS. Six versions of Windows OSs are targeted: Windows NT4 Workstation with SP6, Windows 2000 Professional with SP4, Windows XP Professional with SP1, Windows NT4 Server with SP6, Windows 2000 Server with SP4 and Windows 2003 Server. In the rest of this chapter, Windows 2000 Professional and Windows NT4 Workstation will be referred to as Windows 2000 and Windows NT4 respectively. Four Linux OSs (Debian distribution) are targeted: Linux 2.2.26, Linux 2.4.5, Linux 2.4.26 and Linux 2.6.6. Each of them is a revision of one of the stable versions of Linux (2.2, 2.4, 2.6). Table 3 summarizes the number of system calls targeted by the benchmark experiments carried out along with the number of corresponding parameters and the number of experiments for each OS.

	Windows family						Linux family			
	W- NT4	W- 2000	W- XP	W- NT4S	W- 2000S	W- 2003S	L- 2.2.26	L- 2.4.5	L- 2.4.26	L- 2.6.6
# System Calls	25	27	26	25	27	27	16	16	16	17
# Parameters	53	64	64	53	64	64	38	38	38	44
# Experiments	418	433	424	418	433	433	206	206	206	228

Table 3: Number of system calls, corrupted parameters and experiments for each OS, using PostMark

OS robustness is given in Figure 4. It shows that all OSs of the same family are equivalent, which is in conformance with our previous results, related to Windows using TPC-C Client [Kalakech *et al.* 2004b]. It also shows that none of the catastrophic outcomes (*Panic* or *Hang* OS states) occurred for all Windows and Linux OSs. Linux OSs notified more error codes (59-67%) than Windows (23-27%), while more exceptions were raised with Windows (17-22%) than with Linux (8-10%). More no-signaling cases have been observed for Windows (55-56%) than for Linux (25-32%). In [Shelton *et al.* 2000] it was observed that on the one

hand Windows 95, 98, 98SE and CE had a few Catastrophic failures and on the other hand Windows NT, Windows 2000 and Linux are more robust and did not have any Catastrophic failures, as in our case.

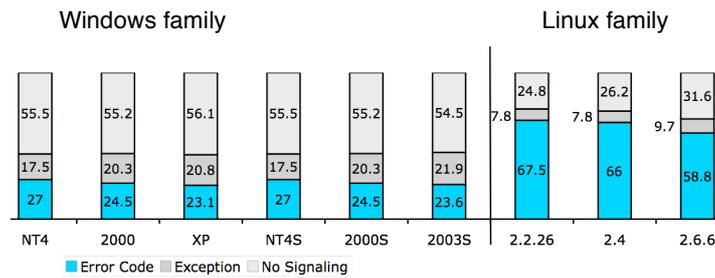


Figure 4: OS Robustness (%), using PostMark

The reaction time is given in Figure 5. Globally, Windows OSs have shorter response times than Linux OSs. The standard deviation is significantly larger than the average for all OSs. Except for the two revisions of Linux 2.4, τ_{reac} is always larger than T_{reac} , the reaction time in the presence of faults. This can be explained by the fact that after parameter corruption, the OS detects the anomaly in almost 45% of cases for Windows and 75% of cases for Linux, and stops system call execution, returns an error code or notifies an exception.

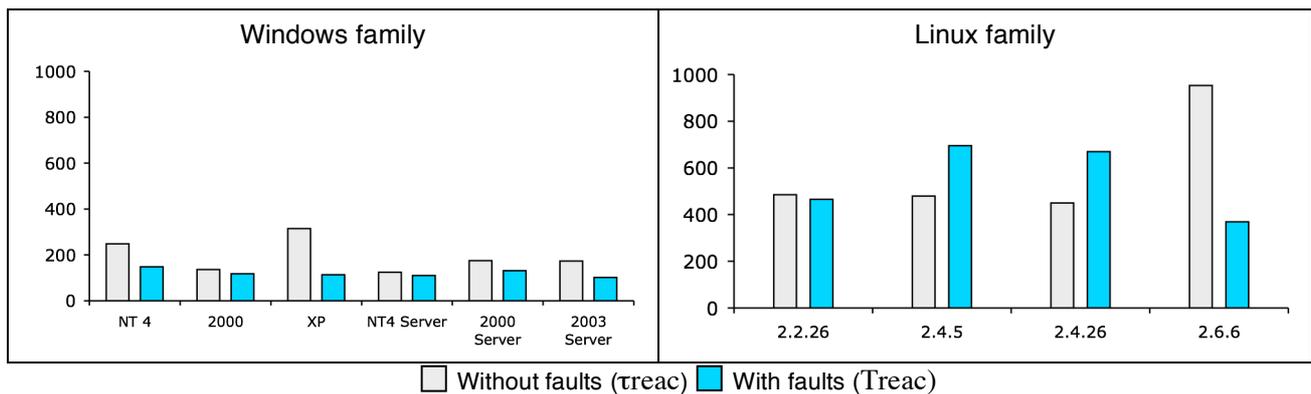


Figure 5: Reaction time (in micro seconds), using Postmark

Note that for the Windows family, Windows XP has the lowest reaction time in the presence of faults and for the Linux family, Linux 2.6.6 has the lowest reaction time. For Linux 2.6.6, we notice that τ_{reac} is almost two times larger than for the other revisions. A detailed analysis of the results showed that this is due to one system call, `execve`, for which the execution time is 15000 μs for Linux 2.6.6 and 6000 μs for other versions.

The restart times are shown in Figure 6. The average restart time without faults, τ_{res} , is always lower than the average restart time with faults (T_{res}), but the difference is not significant. Linux seems to be globally faster (71-83s) than Windows (74-112s). However, if we consider only OS versions introduced in the market after 2001, the other OSs rank as follows: Linux 2.2.26 (71s), Windows XP (74s), Windows 2003 server (77s), Linux 2.4.5 (79s), Linux 2.6.6 (82s), Linux 2.4.26 (83s).

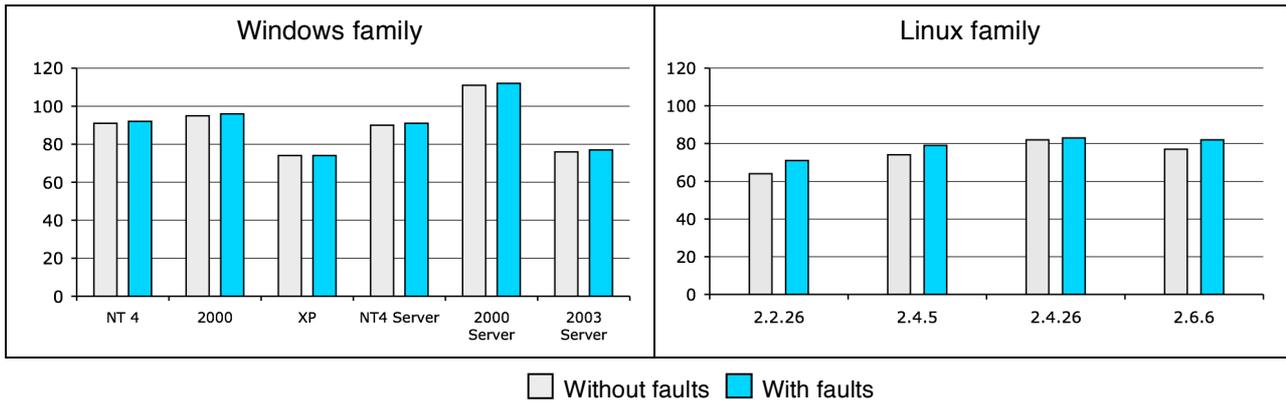


Figure 6: Restart time (in seconds), using Postmark

Concerning Linux family, we note that the restart time increases with new versions or revisions, except for Linux 2.6.6. This progression is due to the increasing size of kernels with the version evolution. The exception of Linux 2.6.6 is justified by the fact that the Linux kernel was restructured in its version 2.6.

4. JVM Dependability Benchmark Implementation and Results

Java Virtual Machine (JVM) is a software layer between the OS and Java applications, allowing applications in Java language to be platform independent. The specifications of the virtual machine [Lindholm & Yellin 1999] are independent from the hardware platform but each platform requires a specific implementation. The benchmark based on JVM has been applied to three Windows versions (NT, 2000 and XP) and to the four Linux versions considered in the previous section. In this benchmark, JVM is solicited through a small program allowing to display «Hello World» on the screen. This program activates 76 system calls with parameters for Windows family and 31 to 37 system calls with parameters for Linux Family, as indicated in Table 4. These system calls are intercepted and corrupted using the corruption technique presented in Section 2.3, which leads to the number of experiments indicated in the last line for each OS.

	Windows family			Linux family			
	W- NT4	W- 2000	W- XP	L- 2.2.26	L- 2.4.5	L- 2.4.26	L- 2.6.6
# System Calls	76	76	76	37	32	32	31
# Parameters	216	214	213	86	77	77	77
# Experiments	1285	1294	1282	457	408	408	409

Table 4: Number of system calls, corrupted parameters and experiments for each OS, using JVM

Robustness is given in Figure 7. As for PostMark workload, no Hang or Panic states have been observed. It can be noticed that the three Windows versions are equivalent, as well as the Linux versions.

Comparison with Figure 4 shows that the robustness of each family is the same using PostMark or JVM workloads (within 5% discrepancy). Further more, we have observed the same robustness for Windows versions using TPC-C as workload in our previous work [Kalakech *et al.* 2004a]. The three workloads solicit different numbers of systems calls and only some of them are the same. Nevertheless they lead to the same robustness.

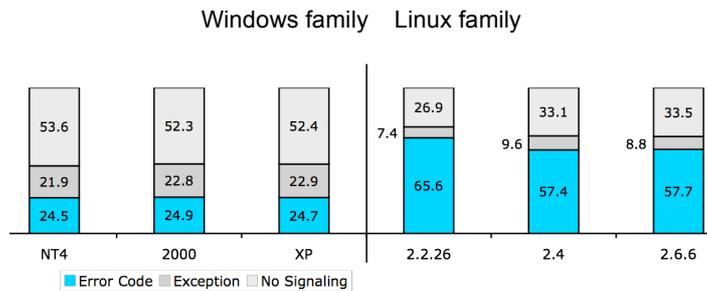


Figure 7: Robustness measures with JVM

The reaction times in the presence of faults (and without fault) are given in Figure 8. Note that for the Windows family, XP has the lowest reaction time, and for the Linux family, 2.6.6 has the lowest one. However, the reaction times of Windows NT and 2000 are very high. A detailed analysis showed that the large response time for Windows NT and 2000 are mainly due to system calls LoadLibraryA, LoadLibraryExA and LoadLibraryEXW. Not including these system calls when evaluating the average of the reaction time in the presence of faults leads respectively to 388 μ s, 182 μ s and 205 μ s for NT4, 2000 and XP. For Linux, the extremely high values of the reaction times without faults are due to two system calls (sched_getparam and sched_getscheduler). Their execution times are significantly larger without fault than in the presence of faults. A detailed analysis of the results showed that for these two system calls, most of the corruption experiments ended with an error code (SEr). Thus, we assume that the system calls were abandoned after an early anomaly detection by the OS. Also for Linux, the reaction times in the presence of faults are relatively high. This is due to three system calls (execve, getdents64 and nanosleep). Not including the reaction times associated with these system calls leads respectively to a Treac of 88 μ s, 241 μ s, 227 μ s and 88 μ s for the 2.2.26, 2.4.5, 2.4.26 and 2.6.6 versions.

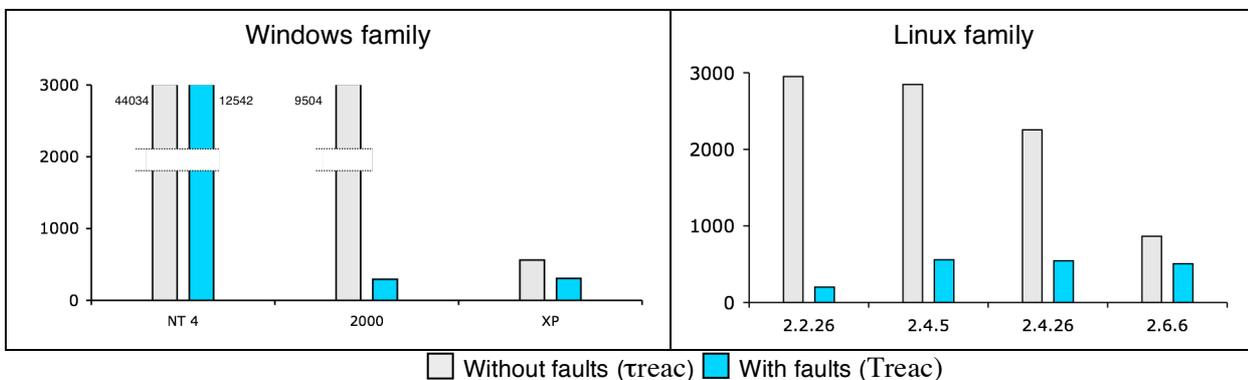


Figure 8: OS reaction time (in micro seconds), using JVM

The restart times are given in Figure 9. As for PostMark workload, the average restart time without faults, tres, is always lower than the benchmark restart time (in the presence of faults), Tres, but the difference is not significant. The standard deviation is very large for all OSs. Linux 2.2.26 and Windows XP have the lowest restart time (71 seconds, in the absence of fault) while Windows NT and 2000 restart times are around 90 seconds and those of Linux versions 2.4.5, 2.4.26 and 2.6.6 are around 80 seconds.

It is interesting to note that the order of the OSs considered is the same for PostMark and for JVM, except for Windows NT and 2000 (which have the highest restart times). Indeed, the only change concerns Windows 2000 whose restart time is slightly decreased for JVM, making it better than Windows NT.

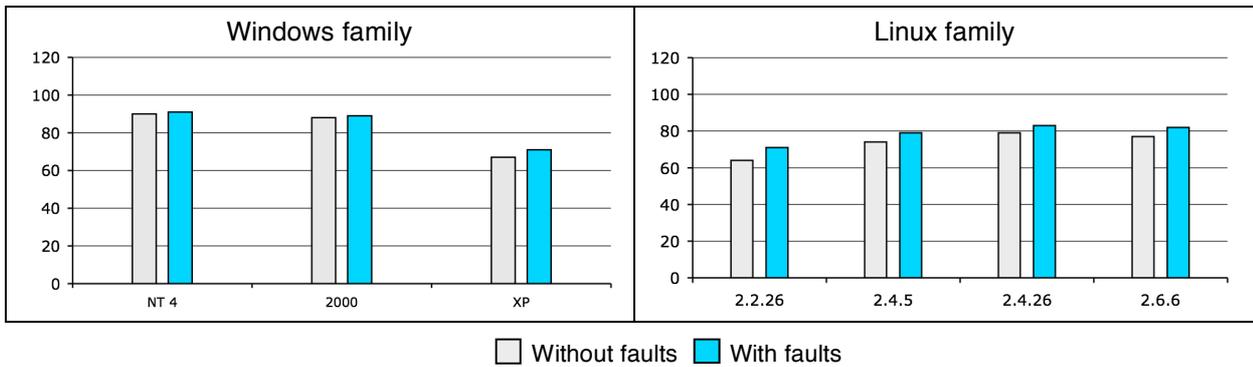


Figure 9: OS restart time (in seconds), using JVM

5. Results Refinement

The benchmark temporal measures are refined to provide more insights into those presented in Sections 3 and 4. We first consider PostMark, then JVM. For each of them, we mainly detail the temporal measures.

5.1. PostMark

5.1.1 Reaction time

Table 5 presents the detailed reaction times with respect to OS outcomes after execution of corrupted system calls (Error Code, Exception and No Signaling). Thus, three average times are added to detail Treac: TSEr, TSXp (the times necessary to return respectively an error code or an exception) and TSNS (the execution time of the corrupted system call, in case of no-signaling state).

For Windows family, it can be seen that for versions 2000, 2000 Server, XP and 2003 Server, returning an error code takes less time than notifying an exception. This can be explained by the fact that when returning an error code, tests are carried out on the parameter values at the beginning of the system call code and the system call is abandoned, while the exceptions are raised from a lower level of the system under benchmarking. Nevertheless, in the cases of Windows NT4 and NT4 Server, TSEr is higher than TSXp. The cause of this anomaly lies in the long time necessary to GetCPIInfo system call to return an error code when its first parameter is corrupted.

Concerning Linux family, the averages presented in this table do not take into account `execve` system call execution time. Cells in grey correspond to high values of the standard deviation and are commented hereafter. We notice the high values of TSNS corresponding to the two revisions of version 2.4, compared to the two other versions. The very high standard deviation suggests a large variation around the average, which is confirmed in Figure 10 that gives the OS reaction time for all system calls leading to the no-signaling state for all Linux OSs. We can clearly see that for Linux 2.4 the average time necessary for executing `mkdir` is more than 10 times larger than for all other system calls. We have noticed that `mkdir` has an extremely long execution time when its second parameter (which corresponds to the permissions to apply on the newly created folder) is corrupted.

Windows Family

	NT4		2000		XP		
	Avg.	S. Dev.	Avg.	S. Dev.	Avg.	S. Dev.	
Treac	148 μ s	219 μ s	118 μ s	289 μ s	114 μ s	218 μ s	
TSEr	45 μ s	107 μ s	34 μ s	61 μ s	45 μ s	118 μ s	
TSXp	40 μ s	15 μ s	37 μ s	15 μ s	50 μ s	96 μ s	
TSNS	234 μ s	437 μ s	186 μ s	375 μ s	168 μ s	265 μ s	
		NT4 Server		2000 Server		2003 Server	
Treac	110 μ s	221 μ s	131 μ s	289 μ s	102 μ s	198 μ s	
TSEr	41 μ s	66 μ s	29 μ s	33 μ s	25 μ s	61 μ s	
TSXp	35 μ s	15 μ s	37 μ s	15 μ s	48 μ s	20 μ s	
TSNS	166 μ s	280 μ s	210 μ s	396 μ s	156 μ s	252 μ s	

Linux Family

	2.2.26		2.4.5		2.4.26		2.6.6	
Treac	167 μ s	300 μ s	466 μ s	2276 μ s	425 μ s	2055 μ s	93 μ s	12 μ s
TSEr	208 μ s	361 μ s	92 μ s	105 μ s	84 μ s	6 μ s	91 μ s	10 μ s
TSXp	88 μ s	5 μ s	91 μ s	8 μ s	91 μ s	8 μ s	106 μ s	13 μ s
TSNS	85 μ s	5 μ s	1545 μ s	4332 μ s	1405 μ s	3912 μ s	91 μ s	11 μ s

Table 5: Detailed reaction time, using PostMark

Also, a very large average time to return an error code is observed for Linux 2.2.26, with a high standard deviation. Figure 11 details the times necessary to return error codes for Linux system calls. It is clear that these times are very similar except for unlink system call in Linux 2.2.26, which explains the high TSEr of Linux 2.2.26 compared to the other versions. After discarding the exceptional values corresponding to `execve`, `mkdir` and `unlink` system calls, the average reaction times T_{reac} of the four targeted Linux OSs become very close. The largest difference is of 8μ s. Also, the average reaction times with respect to OS outcomes after execution of corrupted system calls (TSEr, TSXp, TSNS) become very close. The largest difference is of 18μ s. Furthermore, τ_{reac} and T_{reac} become very close.

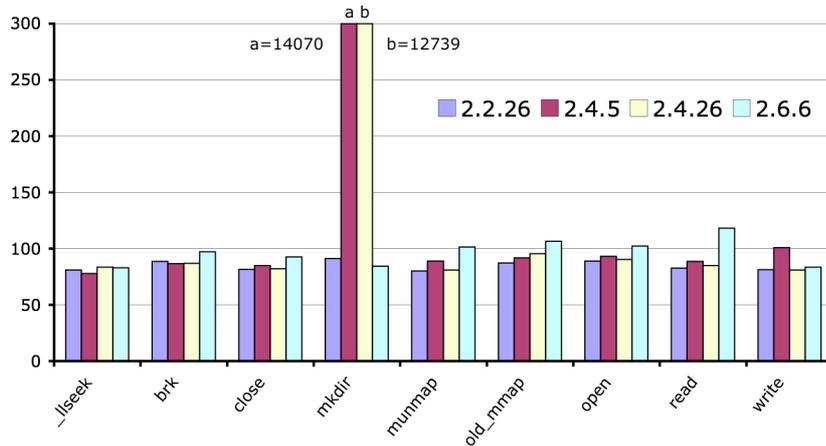


Figure 10: Linux reaction time in case of SNS (in micro seconds), using PostMark

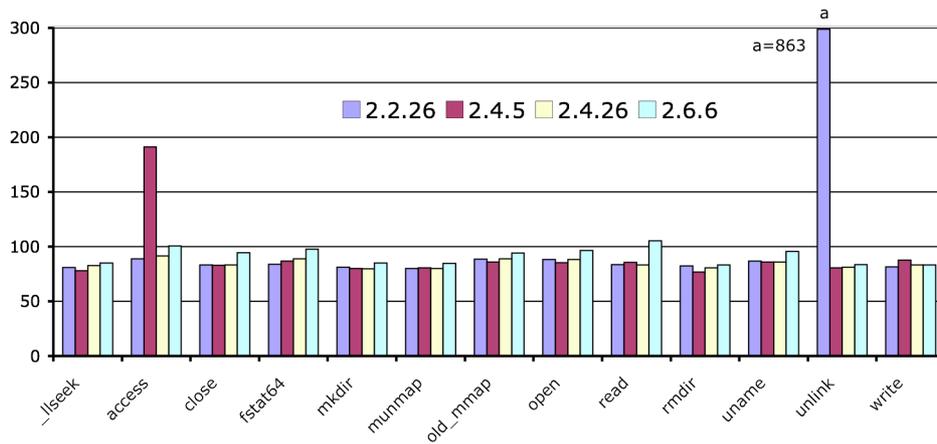


Figure 11: Linux reaction time in case of SER (in micro seconds), using PostMark

5.1.2 Restart time

Detailed analyses show that all OSs of the same family have similar behavior and that the two families exhibit very different behaviors.

For Windows, there is a correlation between the restart time and the state of the workload at the end of the experiment. When the workload is completed, the average restart time is statistically equal to the restart time without parameter substitution. On the other hand, the restart time is larger and statistically equal for all experiments with workload abort/hang. This is illustrated in Figure 12 in case of Windows NT, 2000 and NT. For example, the average restart time in case of workload completion is 73 s and 80 s in case of workload abort/hang, for Windows XP.

Linux restart time is not affected by the workload final state. Detailing Linux restart times shows high values appearing periodically. These values correspond to a “check-disk” performed by the Linux kernel every 26 Target Machine restarts. This is illustrated for Linux 2.2.26 and 2.6.6 in Figure 13, and induces an important standard deviation on this measure. Also, it is interesting to note that the check-disk duration decreases with the version evolution, while the regular restart time increases. It seems natural for the time needed to complete a check-disk to decrease while the Linux kernel evolves. The increase of the regular restart time may be due to the increasing size of Linux kernels.

5.2. JVM

5.2.1 Reaction time

Similarly to Table 5, Table 6 presents the detailed reaction times with respect to OS outcomes after execution of corrupted system calls (Error Code, Exception and No Signaling).

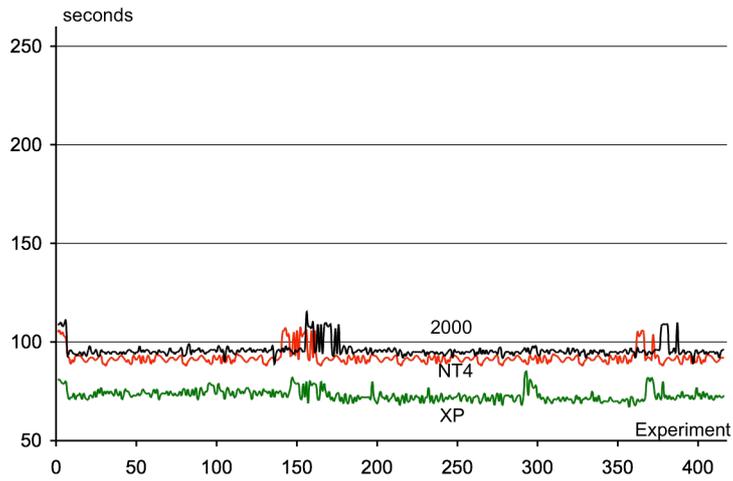


Figure 12: Detailed Windows restart time, using PostMark

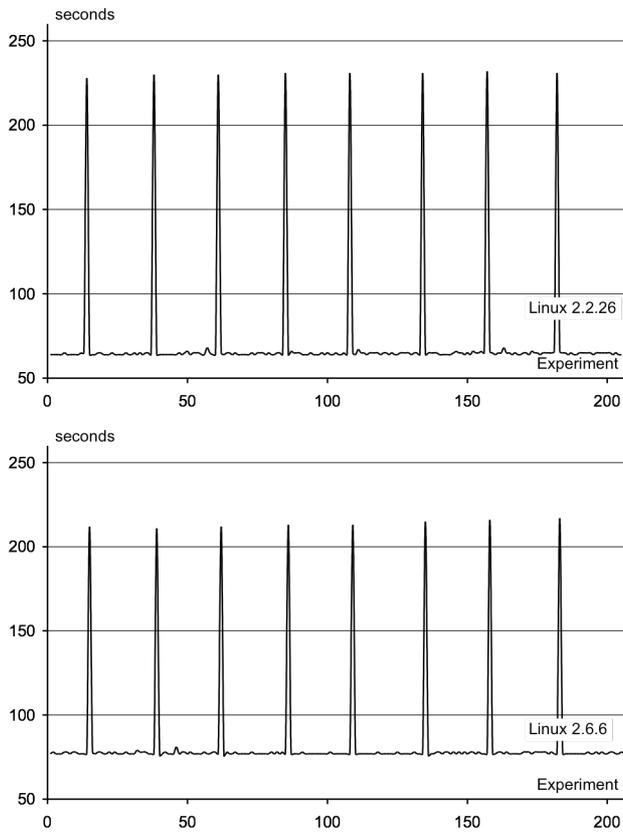


Figure 13: Detailed Linux restart time, using PostMark

Windows Family

	NT4		2000		XP	
	Avg.	S. Dev.	Avg.	S. Dev.	Avg.	S. Dev.
Treac	388 μ s	3142 μ s	190 μ s	451 μ s	214 μ s	483 μ s
TSEr	298 μ s	3006 μ s	47 μ s	110 μ s	51 μ s	103 μ s
TSXp	424 μ s	3862 μ s	84 μ s	168 μ s	98 μ s	209 μ s
TSNS	417 μ s	2858 μ s	307 μ s	588 μ s	344 μ s	625 μ s

Linux Family

	2.2.26		2.4.5		2.4.26		2.6.6	
	Avg.	S. Dev.	Avg.	S. Dev.	Avg.	S. Dev.	Avg.	S. Dev.
Treac	88 μ s	85 μ s	241 μ s	1479 μ s	227 μ s	1438 μ s	88 μ s	26 μ s
TSEr	90 μ s	101 μ s	79 μ s	6 μ s	84 μ s	30 μ s	86 μ s	8 μ s
TSXp	87 μ s	7 μ s	85 μ s	8 μ s	87 μ s	8 μ s	98 μ s	15 μ s
TSNS	84 μ s	6 μ s	572 μ s	2545 μ s	523 μ s	2545 μ s	89 μ s	43 μ s

Table 6: Detailed reaction time, using JVM

For Windows family, the averages presented in this table do not take into account LoadLibraryA, LoadLibraryExA and LoadLibraryExW system call execution times. As in the case of the use of the PostMark workload, we notice that for Windows 2000 and Windows XP TSEr is higher than TSXp, which is higher than TSNS. The standard deviations for these measures are rather small. On the other hand, for Windows NT4 TSEr is lower than TSXp. Figure 14 details the times necessary to raise exceptions for Windows system calls. It is clear that these times are similar except for the system call FindNextFileW. The execution time of this system call is greater than 13400 μ s for Windows NT4 while for the other Windows OSs it is smaller than 75 μ s. It is noteworthy that the execution time of this system call is also the cause for the large values for TSE and TSXp in the case of NT4.

Concerning Linux family, the averages presented in this table do not take into account execve, getdents64 and nanosleep system call execution times. As in the case of the use of the PostMark workload, we notice the high values of TSNS corresponding to the two revisions of version 2.4, compared to the two other versions. The very high standard deviation suggests a large variation around the average, which is confirmed in Figure 15 that gives the OS reaction time for all system calls leading to the no-signaling state for all Linux OSs. As in the case of the use of PostMark workload, for Linux 2.4 the average time necessary for executing mkdir is more than 10 times larger than for all other system calls.

After discarding the exceptional values corresponding to execve, getdents64, nanosleep and mkdir system calls, the average reaction times Treac of the four targeted Linux OSs become very close. The largest difference is of 15 μ s. Also, the average reaction times with respect to OS outcomes after execution of corrupted system calls (TSEr, TSXp, TSNS) become very close. The largest difference is of 28 μ s. Furthermore, τ_{reac} and Treac become very close (if we discard the executions times for sched_getparam and sched_getscheduler system calls).

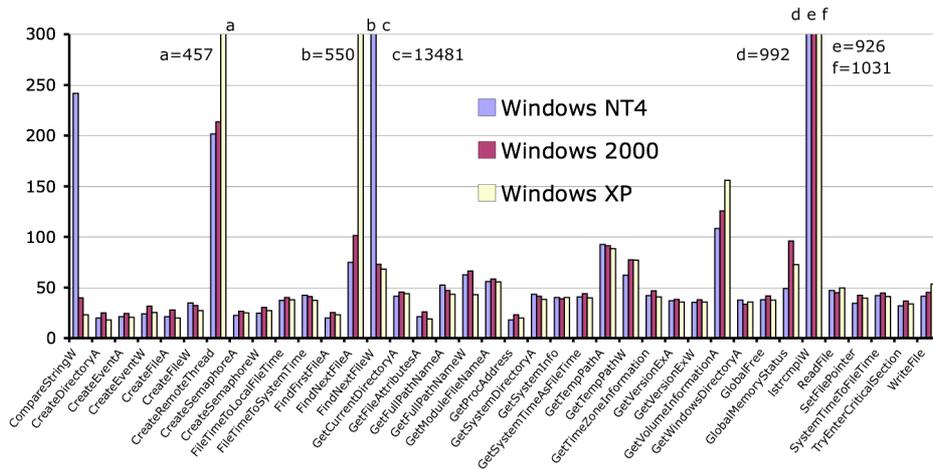


Figure 14: Windows reaction time in case of SXP (in microseconds), using JVM

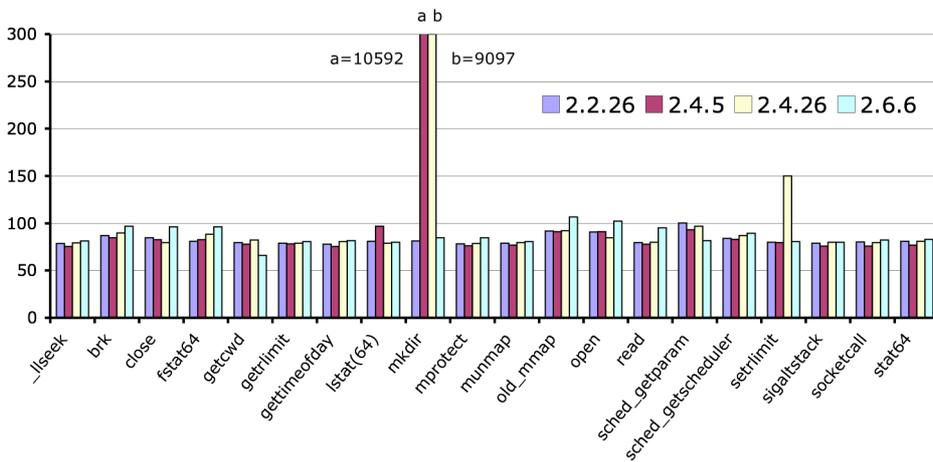


Figure 15: Linux reaction time in case of SNS (in microseconds), using JVM

5.2.2 Restart time

As in the case of the use of PostMark workload, for Windows, there is a correlation between the restart time and the state of the workload at the end of the experiment. When the workload is completed, the average restart time is statistically equal to the restart time without parameter substitution. On the other hand, the restart time is larger and statistically equal for all experiments with workload abort/hang. This is illustrated in Figure 16. For example, for Windows XP, the average restart time in case of workload completion is 68 s and 82 s in case of workload abort/hang.

As in the case of PostMark workload, Linux restart time is not affected by the workload final state. Detailing the restart times shows high values appearing periodically, due to a “check-disk” performed by the kernel every 26 Target Machine restarts. This is illustrated for Linux 2.2.26 and 2.6.6 in Figure 17. Also, it is noteworthy that the check-disk duration decreases with the version evolution, while the regular restart time increases.

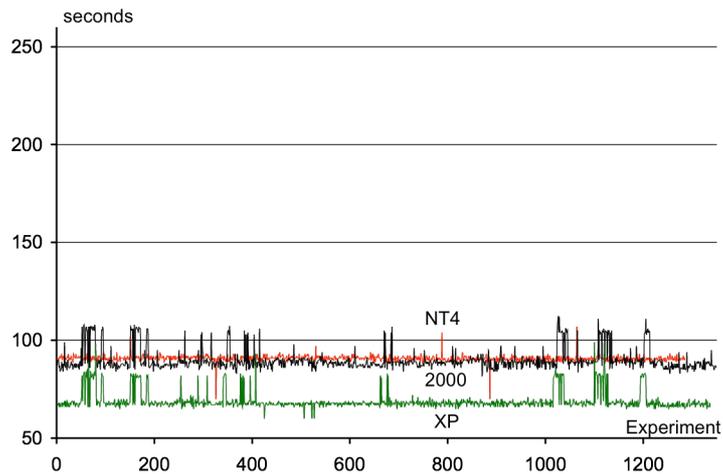


Figure 16: Detailed Windows restart time, using JVM

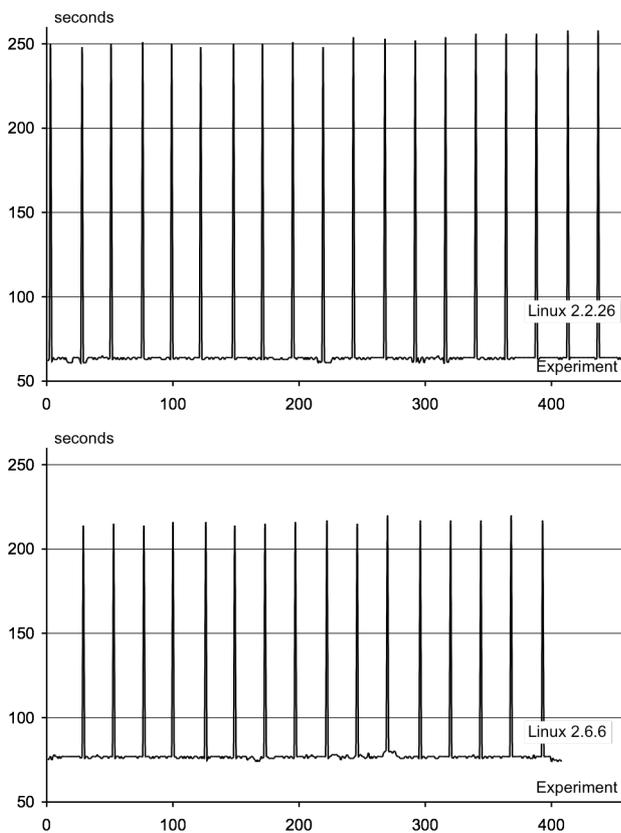


Figure 17: Detailed Linux restart time, using JVM

5.3. PostMark and JVM

OS robustness of all OSs with respect to PostMark and JVM workloads is synthesized in Table 7. It shows that none of the catastrophic outcomes (*Panic* or *Hang* OS states) occurred for all Windows and Linux OSs. It also shows that Linux OSs notified more error codes (57%-67%) than Windows (23%-27%), while more exceptions were raised with Windows (17%-25%) than with Linux (7%-10%). More no-signaling cases have been observed for Windows (52%-56%) than for Linux (25%-33%). For Linux family, Linux 2.2.26 seems to have the most robust behavior (the smallest proportion of no-signaling) while for Windows family, the differences between OSs are too small to differentiate them clearly.

	PostMark			JVM		
	SEr	SXp	SNS	SEr	SXp	SNS
Linux 2.2.26	67.5%	7.8%	24.8%	65.6%	7.4%	26.9%
Linux 2.4.x	66.0%	7.8%	26.2%	57.1%	9.8%	33.1%
Linux 2.6.6	58.8%	9.7%	31.6%	57.7%	8.8%	33.5%
Windows NT4	27.0%	17.5%	55.5%	24.5%	21.8%	53.6%
Windows 2000	24.5%	20.3%	55.2%	24.8%	22.8%	52.3%
Windows XP	23.1%	20.7%	56.1%	24.7%	22.8%	52.4%

Table 7: Linux and Windows robustness using PostMark and JVM

The reaction times in the presence of faults are globally smaller for Windows family OSs than for Linux family OSs (if we exclude a few systems calls for which the reaction time is exceptionally very long). We have noticed that the execution times of a minority of system calls can have important consequences on the mean reaction time values. For instance, the execution time of mkdir system call for Linux 2.4 biases the Treac measure. High standard deviations on this measure are due to a minority of system calls with very large reaction time compared to the majority. If we discard these exceptional values, Treac and τ_{reac} become very close. Moreover, Treac for Windows and Linux families become close.

The restart times in the presence of faults (T_{res}) are always higher than the restart times without faults (τ_{res}). The lowest T_{res} was observed for Linux 2.2.26 and Windows XP (71 s). For Windows family, the restart time is higher in case of workload abort/hang both when using PostMark and JVM as workloads. For Linux family, the standard deviation on this measure is high because of check-disks performed every 26 Target Machine restarts. We have noticed that check-disks take a longer time to perform when using JVM as workload than when using PostMark. The regular restart time is the same both when using PostMark and JVM.

The ordering of the OSs with respect to the restart time is the same for Postmark and JVM except for Windows 2000, for which the restart time is better than Windows NT using JVM while it is worse when using PostMark.

6. Benchmark Properties

In order to gain confidence in dependability benchmark results, one has to check that the key properties are fulfilled. These properties are addressed successively in the rest of this section. We first define the property then we show what has been achieved to satisfy and check it.

6.1. Representativeness

Representativeness concerns the benchmark measures, the workload and the faultload.

6.1.1 Measures

The measures evaluated provide information on the OS state and temporal behavior after execution of corrupted system calls. We emphasize that these measures are of interest to a system developer (or integrator) for selecting the most appropriate OS for his/her own application. Of course other measures would help.

6.1.2 Workload

We have selected two benchmark workloads whose characteristics, in terms of system calls activated, are detailed hereafter. Nevertheless, the selection of any other workload does not affect the concepts and specification of our benchmark.

PostMark workload activates system calls belonging to functional components: file management, thread management, memory management and system information. Most of system calls belong to the file management functional component (62% for Linux, 48% for Windows). However, a significant amount of system calls belong to the thread management (32% for Linux, 12% for Windows) and to the memory management (8% for Linux, 19% for Windows) functional components. PostMark workload is representative if the OS is used as a file server.

JVM workload activates system calls belonging to various functional components (file management, thread management, memory management, user interface, debugging and handling, inter-process communication). Most of the activated system calls belong to the following components: file management (40% for Linux, 26% for Windows), thread management (24% for Linux, 36% for Windows), memory management (24% for Linux, 11% for Windows). JVM workload insures a fair distribution of system calls with respect to functional components. It is noteworthy that, among all system calls (available in Linux), most of them belong to functional components file, thread and memory management.

6.1.3 Faultload

The faultload is without any doubt the most critical component of the OS benchmark and more generally of any dependability benchmark. Faultload representativeness concerns i) the parameter corruption technique used and ii) the set of corrupted parameters.

Parameter corruption technique

In our previous work [Jarboui *et al.* 2002], performed for Linux, we have used two techniques for system call parameter corruption: the *systematic bit-flip* technique consisting in flipping systematically all bits of the target parameters (i.e., flipping the 32 bits of each considered parameter) and the *selective substitution technique* described in Section 2. This work showed the equivalence of the errors induced by the two techniques. In [Kalakech *et al.* 2004b] we obtained the same robustness for Windows 2000 using the systematic bit-flip technique and the selective substitution technique.

The application of the bit-flip technique requires much more experimentation time compared to the application of selective substitution technique. Indeed, in the latter case, the set of values to be substituted is simply determined by the data type of the parameter (see Section 2), which leads to a more focused set of experiments. We have thus preferred the selective substitution technique for pragmatic reasons: it allows derivation of results that are similar to those obtained using the well-known and accepted bit-flip fault injection technique, with much less experiments. Our benchmark is based on selective substitutions of system call parameters to be corrupted.

Parameters to be corrupted

The selective substitution technique used is composed of a mix of three corruption techniques as mentioned in Section 2: out-of-range data (OORD), incorrect data (ID) and incorrect addresses (IA). Let us denote the faultload used in our benchmarks by *FLO*. To analyze the impact of the faultload, we consider two subsets, including respectively i) IA and OORD only (denoted *FL1*), and ii) OORD only (denoted *FL2*). For each workload (PostMark and JVM), we ran the benchmarks of all OSs considered using successively *FLO*, *FL1* and *FL2*. The results obtained confirm the equivalence between Linux family OSs as well as the equivalence

between Windows family OSs, using the same faultload (FL0, FL1 or FL2). Note that for each OS, its robustness with respect to FL0, FL1 or FL2 is different but the robustness of all OSs of the same family with respect to the same faultload is equivalent. The same results have been obtained in [Kalakech *et al.* 2004b], using TPC-C Client as workload.

The number of substitutions (hence the number of experiments) decreases significantly when considering FL1 and F2. By way of examples, Table 8 gives the number of experiments for Windows NT4 and Linux 2.4 for PostMark and Figure 18 shows the robustness of Windows NT4, 2000 and XP with respect to FL1 and FL2, for PostMark. (robustness with respect to FL0 is given in Figure 3).

	ID	IA	OORD	# experiments, PostMark Windows NT4	# experiments, PostMark (Linux 2.4)
FL0	x	x	x	418	206
FL1		x	x	331	135
FL2			x	77	55

Table 8: Faultloads considered

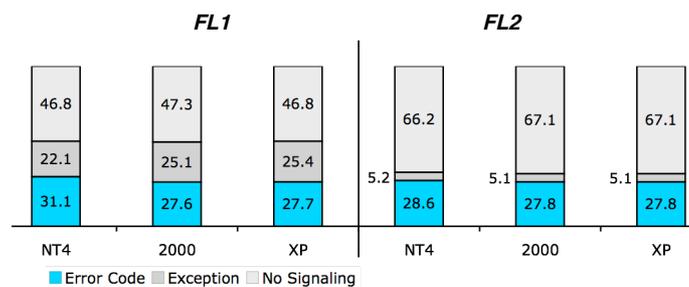


Figure 18: OS Robustness using FL1 and FL2 (%), using PostMark

Further validation concerning selective substitution

For each parameter type class, we performed a sensitivity analysis regarding specific values of parameter substitution. This analysis revealed that different random values chosen to substitute the original parameter lead to the same outcome of benchmark experiments. Hence the benchmark results are not sensitive to the specific values given to the corrupted parameters as substitution values.

Moreover, we checked the representativeness of incorrect data faults. One could argue that the OS is not assumed to detect this kind of faults as the substitution values are inside the validity domain of the parameter type. The analysis of the execution traces corresponding to experiments with incorrect data substitution that led to notification of error codes in the case of Linux, revealed that 88.6% of the faults correspond to out-of-range data in the very particular context of the workload execution. Consequently, the notification of error codes was a normal outcome in these cases. Incorrect data are thus very useful: they can provide a practical way for generating out-of-range data in the execution context. Note that an enormous effort would be needed to analyze all execution contexts for all system calls to define pertinent substitution values for each execution context.

6.2. Repeatability and Reproducibility

The benchmarking of a given system can be based either on an existing benchmark implementation (an existing prototype) or on an existing specification only. Repeatability concerns the benchmark prototype while reproducibility is related to the benchmark specification.

Repeatability is the property that guarantees *statistically equivalent results* when the benchmark is run more than once in the *same environment* (i.e., using the same system under benchmark and the same prototype). This property is central to benchmarking. Our OS dependability benchmark is composed of a series of experiments. Each experiment is run after system restart. The experiments are independent from each other and the order in which the experiments are run is not important at all. Hence, once the system calls to be corrupted are selected and the substitution values defined, the benchmark is fully repeatable. We have repeated all the benchmarks presented three times to check for repeatability.

Reproducibility is the property that guarantees that *another party* obtains statistically equivalent results when the benchmark is implemented from the *same specification* and is used to benchmark the same system. Reproducibility is strongly related to the amount of details given in the specification. The specification should be at the same time i) *general enough* to be applied to the class of systems addressed by the benchmark and ii) *specific enough* to be implemented without distorting the original specification. We managed to satisfy such a tradeoff. Unfortunately, we have not checked explicitly the reproducibility of the benchmark results by developing several prototypes by different people. On the other hand, the results seem to be independent from the technique used to corrupt system call parameters. This makes us confident about reproducibility. However, more verification is still required.

6.3. Portability

Portability concerns essentially the faultload (i.e., its applicability to different OS families).

At the specification level, in order to ensure portability of the faultload, the system calls to be corrupted are not identified individually. We decided to corrupt all system calls of the workloads. This is because OSs from different families do not necessarily comprise the very same system calls as they may have different APIs. However, most OSs feature comparable functional components.

At the implementation level, portability can only be ensured for OSs from the same family because different OS families have different API sets.

Let us consider the case of PostMark as an example, the first prototype developed concerns Windows 2000. It revealed to be portable without modification for Windows 2000 Server and Windows 2003 Server (PostMark activates the same 27 system calls with parameters), and with minor adaptations for the others. One system call (`FreeEnvironmentStringA`) is not activated under Windows NT4, NT4 Server and XP and another system call (`LockResource`) is not activated under NT4 and NT4 Server. In these cases, the system calls that are not activated are dropped from the substitution values database.

For Linux, the prototype revealed to be portable across all OSs except the interceptor Strace that is kernel-dependent. Consequently, we used one version of Strace for Linux 2.2 and 2.4 and another version for Linux 2.6. Also, PostMark activates the same system calls for Linux 2.2.26 and 2.4 while it activates a supplementary system call (`mmap2`) for Linux 2.6.6. Consequently, we added this system call to the set of activated system calls and an entry in the substitution values database.

6.4. Cost

Cost is expressed in terms of effort required to develop the benchmark, run it and obtain results. These steps require some effort that is, from our point of view, relatively affordable. In our case, most of the effort was spent in defining the concepts, characterizing the faultload and studying its representativeness. The implementation of the benchmark itself was not too time consuming.

Let's first consider PostMark, then JVM (which benefited a lot from the PostMark benchmarks as all benchmark components did exist and we had only to adapt them).

For PostMark, the benchmark implementation and running took us less than one month for each OS family, spread as follows:

- The installation of PostMark took one day both for Windows and Linux.
- The implementation of the different components of the controller took about two weeks for each OS family, including the customization of the respective interceptors (Detours and Strace).
- The implementation of the faultload took one week for each OS family, during which we have i) defined the set of substitution values related to each data type and ii) created the database of substitution values. Both databases are portable on OSs belonging to their family (one database for Windows family and one database for Linux family). However, small adaptations were necessary (see Section 6.3).
- The benchmark execution time for each OS is less than two days.

The duration of an experiment with workload completion is less than 3 minutes (including the time to workload completion and the restart time), while it is less than 6 minutes without workload completion (including the watchdog timeout and the restart time). Thus, on average, an experiment lasts less than 5 minutes. The series of experiments of a benchmark is fully automated. Hence, the benchmark execution duration ranges from one day for Linux to less than two days for Windows (25-27 system calls are activated by PostMark on Windows, while only 16-17 system calls are activated on Linux).

For JVM, the first step consisted in executing JVM for each OS to be benchmarked, to identify system calls activated. The second step was devoted to define, for each system call, the parameters to be corrupted and the exact substitution values, to prepare the database to be used in the Interception/substitution/observation modules. This step took a couple of days for Linux family (activating 31-37 system calls depending on the version considered) and the double for Windows as it activates 76 system calls. Adaptation of the benchmark controller and of the Interception/substitution/observation modules required about one day for each family. The benchmark duration ranges from one day for each Linux OS to less than three days for each Windows OS.

7. Conclusion

We presented the specification of a dependability benchmark for OSs with respect to erroneous parameters in system calls, along with prototypes for two families of OSs, Windows and Linux and for two workloads. These prototypes allowed us to obtain the benchmark measures defined in the specification. We stress that the measures obtained for the different OSs are comparable as i) the same workloads (PostMark and JVM) were used to activate all OSs, ii) the faultload corresponds to similar selective substitution techniques applied to all system calls activated by the workload and iii) the benchmark conduct was the same for all OSs.

Concerning the robustness measure, the benchmark results show that all OSs of the same family are equivalent. They also show that none of the catastrophic states of the OS (*Panic* or *Hang*) occurred for any

of the Windows and Linux OSs considered. Linux OSs notified more error codes than Windows while more exceptions were raised with Windows than with Linux. More no-signaling cases have been observed for Windows than for Linux.

Concerning the OS reaction time, results show that globally Linux reaction time, related to system calls activated by the workload is longer than Windows reaction time. Refinement of this measure revealed a great variation around the average and that a minority of system calls with large execution times dodged the average. When these system calls are not considered, the reaction times of all the OSs of the same family become equivalent.

With respect to the restart time measure, Windows XP and Linux 2.2.26 have the shortest restart times in the presence of faults (71 s). Detailed analysis showed i) a correlation between Windows restart time and the workload final state (in case of workload *hang* or *abort*, the restart time is 10 % higher than in case of workload completion) and ii) that Linux performs a check-disk after each 26 restarts. A restart with a check-disk is three to four times longer than the average.

We validated our benchmark paying a particular attention to representativeness of faultload, and to the properties of repeatability, reproducibility, portability and cost effectiveness of the benchmark.

Acknowledgement

We would like to thank all the DBench colleagues who, through the numerous discussions all over the project, helped us in defining the OS benchmark as it is in this paper. In particular, we are grateful to Jean Arlat who contributed to the OS benchmark based on TPC-C Client.

References

- [Albinet *et al.* 2004] A. Albinet, J. Arlat and J.-C. Fabre, "Characterization of the Impact of Faulty Drivers on the Robustness of the Linux Kernel", in *Int. Conf. on Dependable Systems and Networks*, (Florence, Italy), pp. 867-876, 2004.
- [Arlat *et al.* 2002] J. Arlat, J.-C. Fabre, M. Rodríguez and F. Salles, "Dependability of COTS Microkernel-Based Systems", *IEEE Transactions on Computers*, vol. 51, no. 2, pp. 138-163, February 2002.
- [Brown *et al.* 2002] A. Brown, L. C. Chung and D. A. Patterson, "Including the Human Factor in Dependability Benchmarks", in *Proc. of the 2002 DSN Workshop on Dependability Benchmarking*, (Washington, DC, USA), 2002.
- [Brown & Patterson 2000] A. Brown and D. A. Patterson, "Towards Availability Benchmarks: A Cases Study of Software RAID Systems", in *Proc. 2000 USENIX Annual Technical Conference*, (San Diego, CA, USA), USENIX Association, 2000.
- [Chevochot & Puaut 2001] P. Chevochot and I. Puaut, "Experimental Evaluation of the Fail-Silent Behavior of a Distributed Real-Time Run-Time Support Built from COTS Components", in *Proc. Int. Conference on Dependable Systems and Networks (DSN-2001)*, (Göteborg, Sweden), pp. 304-313, IEEE CS Press, 2001.
- [Chou *et al.* 2001] A. Chou, J. Yang, B. Chelf, S. Hallem and D. Engler, "An Empirical Study of Operating Systems Errors", in *Proc. 18th ACM Symp. on Operating Systems Principles (SOSP-2001)*, (Banff, AL, Canada), pp. 73-88, ACM Press, 2001.
- [DBench] <http://www.laas.fr/DBench>, Project Reports section, project short final report.
- [Durães & Madeira 2002] J. Durães and H. Madeira, "Characterization of Operating Systems Behavior in the Presence of Faulty Drivers through Software Fault Emulation", in *Proc. 2002 Pacific Rim Int. Symp. on Dependable Computing (PRDC-2002)*, (Tsukuba City, Ibaraki, Japan), pp. 201-209, 2002.

- [Gu *et al.* 2004] W. Gu, Z. Kalbarczyk and R. K. Iyer, “Error Sensitivity of the Linux Kernel Executing on PowerPC G4 and Pentium 4 Processors”, in *Int. Conf. on Dependable Systems and Networks*, (Florence, Italy), pp. 887-896, 2004.
- [Hunt & Brubaker 1999] G. Hunt and D. Brubaker, “Detours: Binary Interception of Win32 Functions”, in *3rd USENIX Windows NT Symposium*, (Seattle, Washington, USA), pp. 135-144, 1999.
- [Jarboui *et al.* 2002] T. Jarboui, J. Arlat, Y. Crouzet, K. Kanoun and T. Marteau, “Analysis of the Effects of Real and Injected Software Faults: Linux as a Case Study”, in *Proc. 2002 Pacific Rim Int. Symp. on Dependable Computing (PRDC-2002)*, (Tsukuba City, Ibaraki, Japan), pp. 51-58, IEEE CS Press, 2002.
- [Kalakech *et al.* 2004a] A. Kalakech, T. Jarboui, J. Arlat, Y. Crouzet and K. Kanoun, “Benchmarking Operating System Dependability: Windows 2000 as a case study”, in *Proc. 2004 Pacific Rim Int. Symp. on Dependable Computing (PRDC-2004)*, (Papeete, Tahiti), pp. 261-270, 2004.
- [Kalakech *et al.* 2004b] A. Kalakech, K. Kanoun, Y. Crouzet and J. Arlat, “Benchmarking the Dependability of Windows NT, 2000 and XP”, in *Proc. Int. Conf. on Dependable Systems and Networks (DSN-2004)*, (Florence, Italy), pp. 681-686, 2004.
- [Kanoun *et al.* 2005] K. Kanoun, Y. Crouzet, A. Kalakech, A.-E. Rugina and P. Rumeau, “Benchmarking the Dependability of Windows and Linux using Postmark workloads”, in *Proc. 16th Int. Symposium on Software Reliability Engineering (ISSRE-2006)*, (Chicago, USA), 2005.
- [Kanoun & Crouzet 2006] K. Kanoun and Y. Crouzet, “Dependability Benchmarking for Operating Systems”, in *International Journal of Performance Engineering*, vol. 2, no. 3, pp. 275-287, 2006.
- [Katcher 1997] J. Katcher, *PostMark: A New File System Benchmark*, Network Appliance, www.netapp.com/tech_library/3022.html, N°3022, 1997.
- [Koopman & DeVale 1999] P. Koopman and J. DeVale, “Comparing the Robustness of POSIX Operating Systems”, in *Proc. 29th Int. Symp. on Fault-Tolerant Computing (FTCS-29)*, (Madison, WI, USA), pp. 30-37, IEEE CS Press, 1999.
- [Koopman *et al.* 1997] P. J. Koopman, J. Sung, C. Dingman, D. P. Siewiorek and T. Marz, “Comparing Operating Systems using Robustness Benchmarks”, in *Proc. 16th Int. Symp. on Reliable Distributed Systems (SRDS-16)*, (Durham, NC, USA), pp. 72-79, IEEE CS Press, 1997.
- [Lindholm & Yellin 1999] T. Lindholm and F. Yellin, *The Java Virtual Machine Specification*, Addison-Wesley Professional, 1999.
- [McGrath & Akkerman 2004] R. McGrath and W. Akkerman, *Source Forge Strace Project*, <http://sourceforge.net/projects/strace/>, 2004.
- [Mukherjee & Siewiorek 1997] A. Mukherjee and D. P. Siewiorek, “Measuring Software Dependability by Robustness Benchmarking”, *IEEE Transactions of Software Engineering*, vol. 23 no. 6, pp. 366-376, 1997.
- [Shelton *et al.* 2000] C. Shelton, P. Koopman and K. D. Vale, “Robustness Testing of the Microsoft Win32 API”, in *Proc. Int. Conference on Dependable Systems and Networks (DSN-2000)*, (New York, NY, USA), pp. 261-270, IEEE CS Press, 2000.
- [TPC-C 2002] TPC-C, “*TPC Benchmark C, Standard Specification 5.1, available at <http://www.tpc.org/tpcc/>*” 2002.
- [Tsai *et al.* 1996] T. K. Tsai, R. K. Iyer and D. Jewitt, “An Approach Towards Benchmarking of Fault-Tolerant Commercial Systems”, in *Proc. 26th Int. Symp. on Fault-Tolerant Computing (FTCS-26)*, (Sendai, Japan), pp. 314-323, IEEE CS Press, 1996.
- [Zhu *et al.* 2002] J. Zhu, J. Mauro and I. Pramanick, *R-Cubed (R3): Rate, Robustness, and Recovery - An Availability Benchmark Framework*, Sun Microsystems Laboratories, N°TR-2002-109, 2002.

INVESTIGATING THE EFFICIENCY OF CRYPTOGRAPHIC ALGORITHMS IN ONLINE TRANSACTIONS

C. Lamprecht¹ A. van Moorsel P. Tomlinson N. Thomas

School of Computing Science,
University of Newcastle upon Tyne, UK

Abstract

Web Service security is an important factor for Web Services to gain increased acceptance. This paper presents how message level security is achieved in web services interactions and we evaluate a number of commonly used cryptographic algorithms to determine which are most suitable for the task. In particular we explore whether VeriSign's Trusted Services Integration Kit (TSIK) is a viable option for realising this. Furthermore, through measurement of TSIK as well as of an implementation using Java Cryptography Extensions (JCE), we conclude that TSIK provides an adequate level of security with minimal additional overheads. However, it would benefit from using SHA-256 and IDEA in future releases as well as decreasing algorithm operation time when processing larger messages.

1. Introduction

Since the advent of the World Wide Web, e-commerce has become a major source of interest for both businesses and customers. Since most transactions typically occur over the Internet, in the public domain, much research has been undertaken to use cryptographic algorithms to secure messages during these online transactions. Though emphasis has been primarily placed on the level of security which such algorithms provide, it is also of interest to evaluate their efficiency at doing so. This is particularly relevant in areas such as stock trading or online

bidding where the dynamic nature of the data accessed imposes real-time constraints on the transaction.

More recently Web Services have been met with growing interest from academia as well as industry due to its potential to provide a generic global service oriented network which is flexible enough to cater for individual service needs as well as providing increased interoperability between services. As such, e-commerce could benefit greatly from adopting Web Services technologies.

In this paper we will present three different cryptographic methods typically used to secure messages during online transactions. For each we will consider the available algorithms and conduct a comparative evaluation based on their performance. This will aid in determining suitable cryptographic algorithms for transactions with real-time constraints. We also consider Web Services' claim as a potential solution environment and evaluate Verisign's Trusted Services Integration Kit (TSIK), a hybrid solution based on a set of cryptographic algorithms, with respect to the level of security it provides as well as its efficiency at doing so.

We first discuss, in Section 2, what cryptographic methods are required to secure messages in online transactions and in Section 3 provide a comparative analysis of available algorithms using the Sun Java Cryptography Extensions (JCE) [SunJCE2005] as well as the Cryptix extensions for Java [Cryptix2005]. Section 4 details a comparative evaluation of TSIK's performance and level of security it provides with respect to using the Sun Java Cryptography Extensions. The paper concludes with a summary in Section 5.

¹ Communicating author,
C.J.Lamprecht@ncl.ac.uk

2. Securing Transactions

Online transactions typically require: message integrity to ensure messages are unaltered during transit; message confidentiality to ensure message content remain secret; non-repudiation to ensure that the sending party cannot deny sending the received message; and sender authentication to prove sender identity.

We provide a brief overview of the well-known cryptographic techniques available to achieve the above.

2.1 Symmetric cryptography

Symmetric cryptography tries to ensure message confidentiality by encrypting the message (the plaintext) using a secret key to produce an encrypted version of the message (the cipher text), which is then sent instead of the original message. Message integrity is implicitly provided, as altering the cipher text would result in an illegible decrypted message. 'Symmetric' refers to the fact that the same secret key is required to decrypt the message on the recipient's side. Typical symmetric encryption algorithms include DES, Triple DES, RC2, RC5, Twofish, Blowfish, IDEA and AES. Most symmetric algorithms can operate in two modes, namely Cipher Block Chaining Mode (CBC) or Electronic Codebook Mode (ECB). The former of which is considered more secure as it ensures that encrypting the same plaintext never produces the same cipher text. The main problem in this scheme is the key distribution problem; since the same secret key is used to decrypt the message, one must find a way to securely transport the key from sender to recipient.

2.2 Asymmetric cryptography (public key cryptography)

Asymmetric cryptography provides the same message security guarantees as symmetric cryptography, but additionally provides the non-repudiation guarantee. 'Asymmetric' refers to the fact that different keys are used for encryption and decryption. One key is kept secret ('secret key') and the other is made public ('public key'), and are both unique. The recipient's public key should be used during

the encryption process to ensure message confidentiality as only the recipient has the necessary secret key to decrypt the message. If, however, the message is encrypted using the sender's private key the sender cannot deny sending the message as his private key is unique and is only known to him. Typical asymmetric algorithms include RSA, ElGamal and DSA. Asymmetric cryptography is extremely powerful, but this comes at a cost. Especially for longer messages and keys, it is much slower than its symmetric cryptography counterparts [Adams2003]. This is due in part to the fact that, in order to achieve comparable security, asymmetric keys are generally around an order of magnitude longer than symmetric keys.

2.3 Hashing

Hashing tries to ensure message integrity by producing a condensed version of the message, known as the message digest, which is unique to that message. The hashing algorithm is publicly known and so the recipient can perform the same hash on the received message, to produce another message digest, and compare it to the received digest to assess whether the original message has been altered. Typical hashing algorithms include MD2, MD4, MD5, RIPEMD, SHA-1, SHA-256, SHA-384 and SHA-512. Hashing does not provide confidentiality, non-repudiation or authentication. On its own, hashing does not provide message integrity either as both the hash and the message could be replaced by a third party and so prevent the recipient from detecting the attack. Section 4.1 explains how hashing is utilized to ensure message integrity.

3. Cryptographic Techniques

The following section details a performance evaluation of the most common cryptographic algorithms for each cryptographic technique to determine their suitability for systems with real-time constraints. All experiments were conducted on a 1GHz machine with 256MB RAM running Linux Fedora Core. For each experiment a 1,137 byte plaintext file was used. All results for symmetric and asymmetric algorithms include key generation, algorithm

initialization and message encryption times. The experiments were repeated several times with negligible variance in the results.

3.1 Symmetric cryptography

This section details a comparative performance evaluation of a subset of symmetric encryption algorithms. Sun Java Cryptography Extensions [SunJCE2005] (referred to hereafter as *JCE*) as well as Java Cryptix Libraries [Cryptix2005] (referred to hereafter as *Cryptix*) are used for this purpose. Using Cryptix we furthermore investigate whether either Cipher Block Chaining Mode (CBC) or Electronic Codebook Mode (ECB) boasts a performance advantage. 128 bit key size was used for all algorithms with the exception of DES (56 bits), DESede (Triple DES using 112 bits) and Skipjack (80 bits) as they require fixed key sizes. Unless stated CBC mode was used.

3.1.1 Algorithms

Looking at Figures 1 and 2, the first observation to make is that there are significant differences between the observed durations shown in each graph; JCE took much longer than Cryptix for the same algorithm. The conclusion we draw from this is that the implementation has a large impact on the efficiency of the execution. This is further emphasized when individual algorithms are compared. Naively one would expect that Triple DES would take three times as long as DES. However, this is evidently not the case, being only about 25% slower in Cryptix and only very marginally slower in JCE. Clearly this is influenced by the implementation, and conceivably the Java Virtual Machine optimizations are also playing a part in apparently “speeding up” Triple DES.

The algorithm which consistently performed the best in our evaluation was IDEA. According to Schneier [Schneier1996], IDEA is approximately twice as fast as DES; in our experiments it was closer to three times as fast. Perhaps surprisingly, Blowfish was much slower, only a little better than DES and slower than algorithms such as Skipjack and Serpent. Blowfish was designed to be fast and requires

little memory [Schneier1996], but we did not find this Cryptix distribution particularly efficient in our experimental set up. AES (Rjindael) performed particularly badly in the JCE distribution, but less poorly in the Cryptix distribution. We were unable to satisfactorily explain this difference, except as further evidence of how the implementation of an algorithm can severely impact the actual performance.

What is not evident in these plots is the relative security of the different algorithms. In this respect key length is a good indicator, and so DES and Skipjack may be considered to be potentially less secure than others. Overall therefore it appears that IDEA is the best choice among the symmetric algorithms tested, as it provides adequate security as well as a fast execution time.

3.1.2 Encryption Mode

Figure 3 clearly indicates that neither mode shows a significant performance advantage. It would therefore seem prudent to use CBC mode during message encryption as discussed in Section 2.1.

3.2 Asymmetric cryptography (public key cryptography)

The results presented in Figure 4 were obtained using the standard Java Cryptography Extensions (JCE). The graph shows the average time to generate keys and encrypt 1,137 bytes of data. It can be seen that the RSA algorithm family outperforms that of Diffie-Hellman at all key lengths in our experiments. A key length of 1024 bits is currently considered to be the minimum secure length for both RSA and Diffie-Hellman. Diffie-Hellman with a key length of 1024 was also considered but yielded results 30 times slower than that of its RSA counterpart, and so is not shown for reasons of clarity. DSA also performed well but can only be used for non-repudiation purposes and not for data confidentiality. RSA and Diffie-Hellman are able to support both.

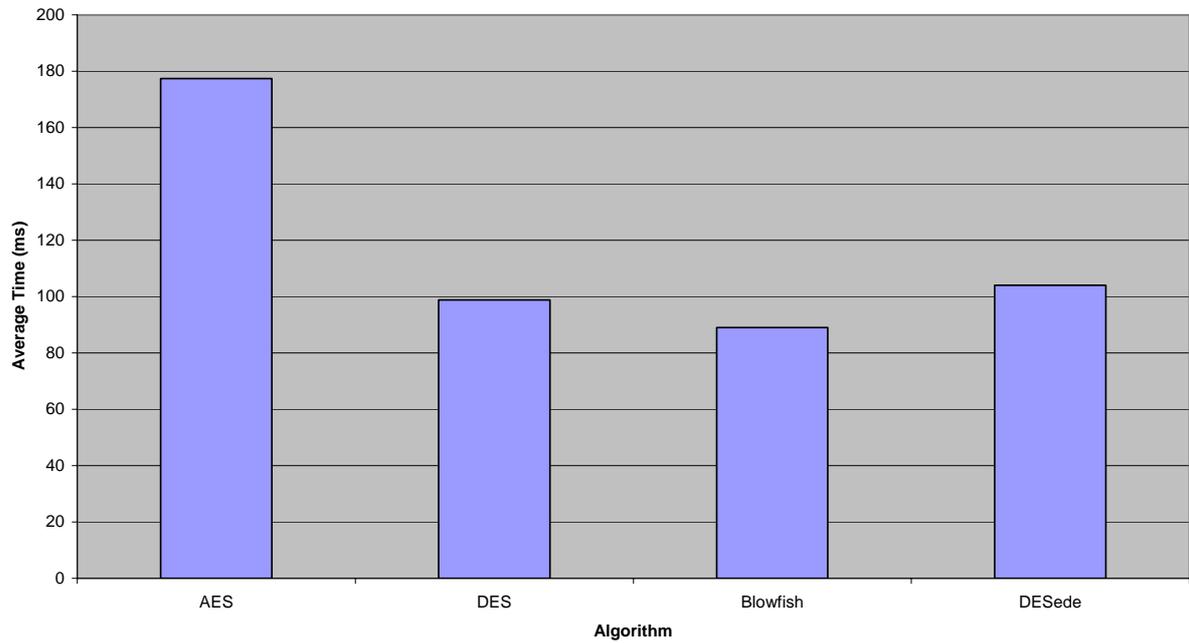


Figure 1: Average time to encrypt a 1137B file using JCE distributions

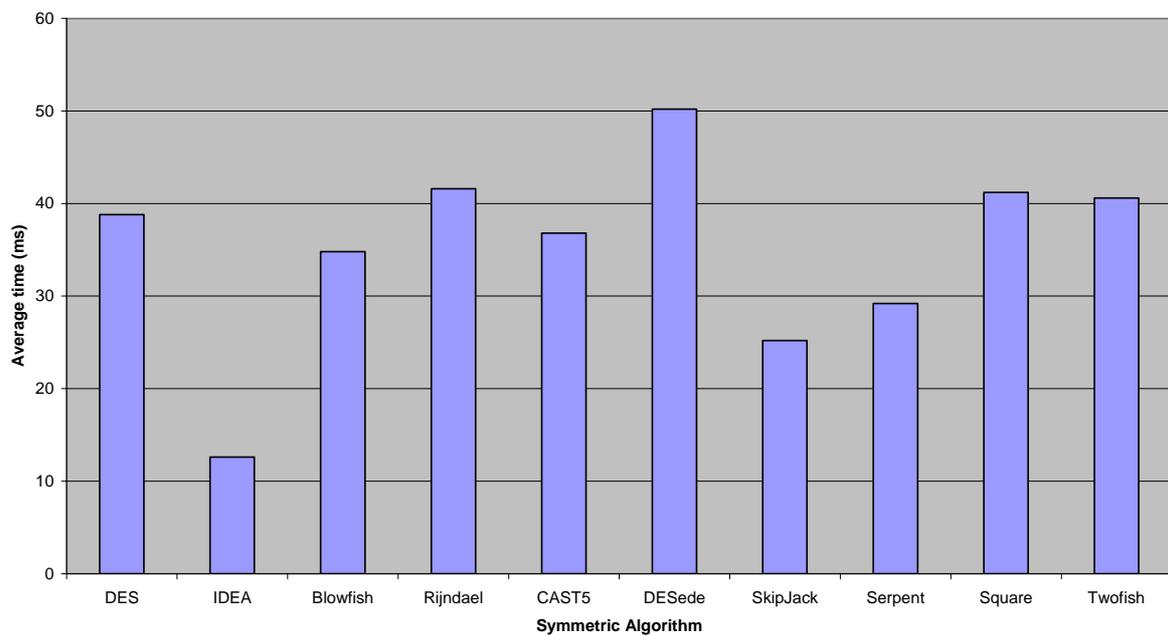


Figure 2: Average time to encrypt a 1137B file using Cryptix distributions

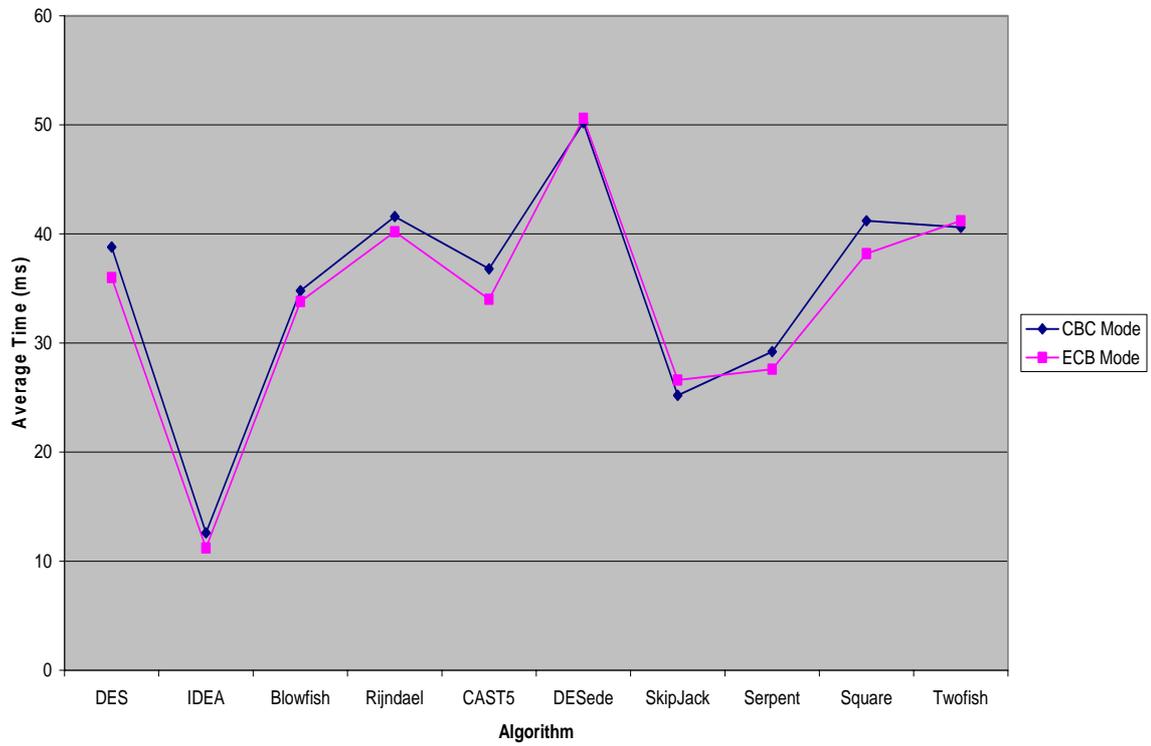


Figure 3: A comparison of symmetric algorithms operating in ECB mode and CBC mode

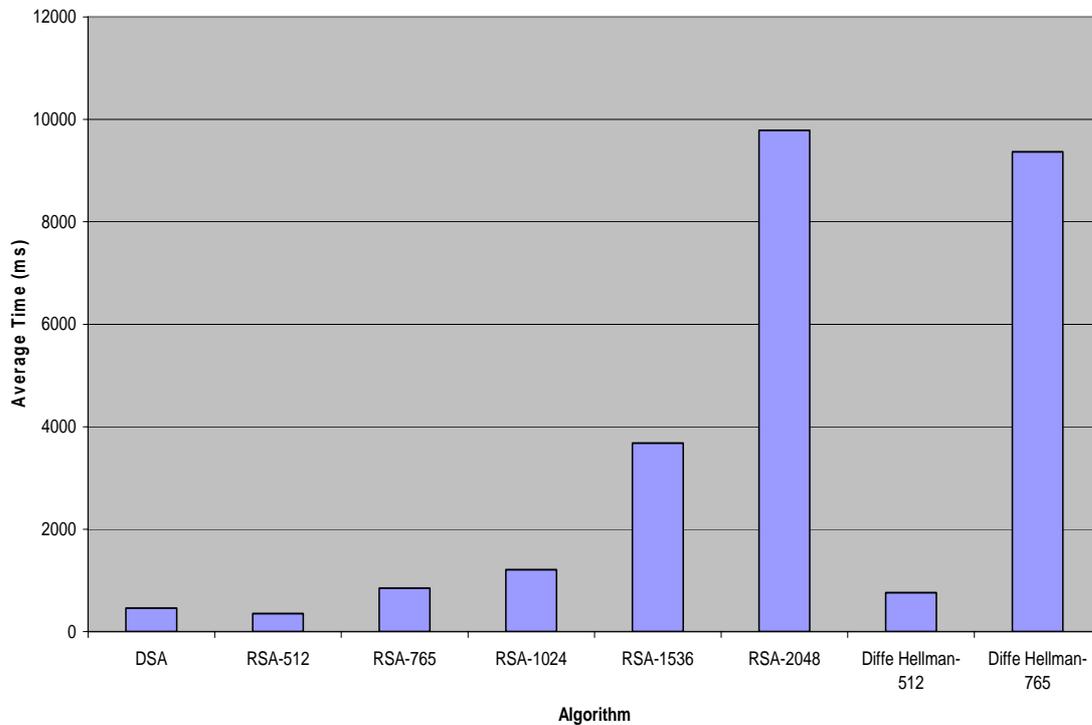


Figure 4: Average time for key generation and encryption using public key algorithms

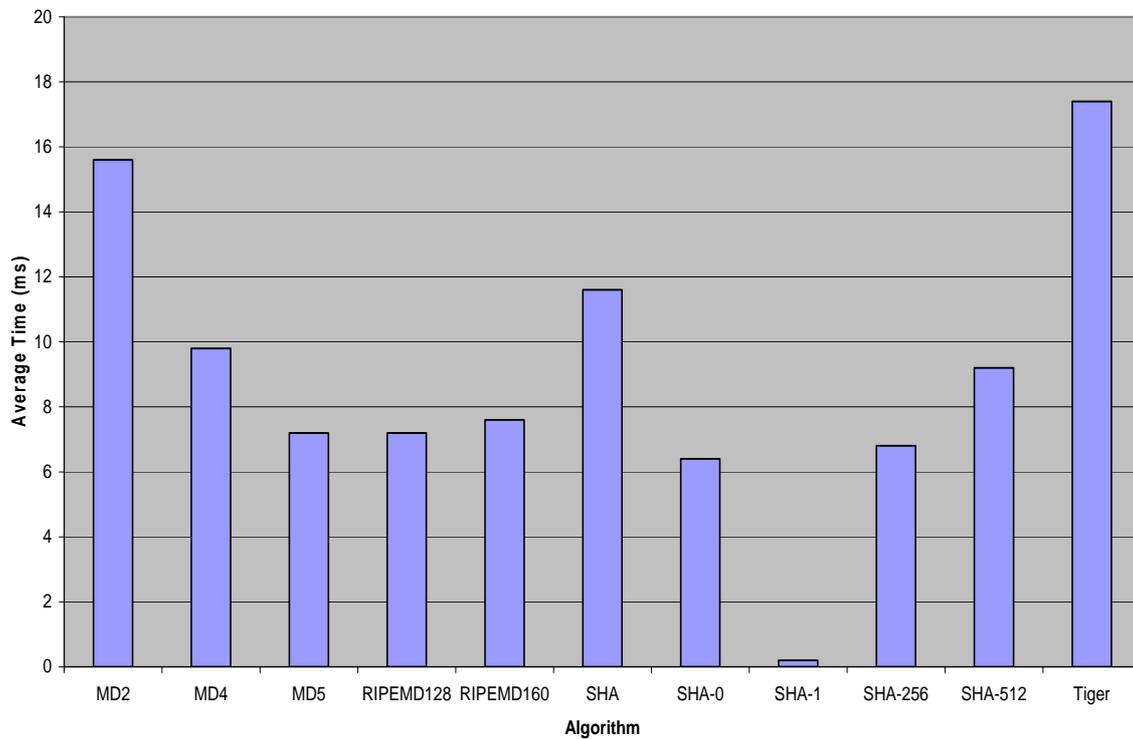


Figure 5: Average time to generate a message digest

3.3 Hashing

Java Cryptix Libraries were used in this experiment. 128 bit key size was used for all MD algorithms, 160 bits for SHA (unless otherwise stated) and 190 bits for Tiger.

As can be seen in Figure 5, SHA-1 significantly outperforms all other considered algorithms. Unfortunately SHA-1 has recently been shown to be less secure than initially anticipated and SHA-256 is currently recommended [Lenstra2005]. RIPEMD with key sizes 128 bits and 160 have been developed to replace the 128 bit MD algorithms. Both RIPEMD algorithms seem to achieve comparable performance to that of SHA-256, though clearly have shorter key sizes and so potentially less secure.

3.4 Summary

The results presented suggest that RSA-1024 and SHA-256 are the most suitable cryptographic algorithms for use during transactions in systems with real-time constraints. Almost any of the symmetric algorithms could be selected, but IDEA was shown to be the fastest in our evaluation.

4. Hybrid system

Web Services are built on open standards to provide a generic way of communication between heterogeneous environments. It therefore shows particular potential to be exploited within the e-commerce domain. In this section we investigate this further. In particular, we consider the combination of

cryptographic algorithms used in VeriSign's Trusted Services Integration Kit (TSIK) and evaluate them based on the level of security they provide as well as their performance and so conclude whether it is a suitable alternative for transactions with real-time constraints. TSIK's performance is evaluated through direct comparison with Java's Cryptography Extensions (JCE).

We therefore first detail the concepts that constitute such a hybrid system in Section 4.1. Section 4.2 analyses one element of the hybrid system, namely asymmetric cryptography, to aid in evaluating the level of security provided by TSIK, as detailed in Section 4.3, as well as understanding the results in Section 4.4. Final conclusions are drawn in Section 4.5.

4.1 System functionality

The hybrid system exhibits the following functionality, in which the techniques detailed in Section 2 are combined to achieve a more effective security solution through signing, verifying, encryption and decryption. They are combined as follows:

The key, in symmetric cryptography, can be securely transported using public key cryptography by encrypting the symmetric key using the receiver's public key. The receiver, and only the receiver, can then first decrypt the symmetric key using his private key and then decrypt the message using the decrypted symmetric key. Also note that only the key, which is relatively short, is encrypted using public key cryptography and so reduces encryption overhead.

The message digest, produced by the hash function, can be encrypted using an asymmetric cryptography algorithm to avoid and interception attack. Thus, if the message digest is encrypted using the sender's private key, only the message can be replaced during transit and not the message digest, since the interceptor does not have the sender's private key to encrypt the new message digest.

Generating a message digest and then encrypting the message digest using a private key is referred to as signing the message.

Decrypting the message digest using the sender's public key, generating a new message digest of the received message and then comparing the digests is called verifying the message. The performance results of these two techniques, among others, are analysed in this paper.

Sender authentication is achieved when the sender's public key is signed by a mutually trusted third party. The receiver can then verify the public key as the third party's public key is trusted.

4.2 RSA [Rivest1978]

Understanding the security implications and performance results in Section 4.3 and Section 4.4 requires a deeper understanding of public key cryptography. In particular RSA, which was developed by Ron Rivest, Adi Shamir and Leonard Adleman in 1977 and is used by VeriSign's TSIK toolkit. We do not explain all the details of RSA, but instead focus on the particular use of RSA in our measurement setup.

4.2.1 The algorithm [Rivest2003, Sun2005]

- Choose 2 large primes p and q such that $pq = N$
- Select 2 integers e and d such that $ed = 1 \text{ mod } \phi(N)$
 - Where $\phi(N) = (p-1)(q-1)$ is the Euler totient function of N

In general, N is called the modulus, e the public exponent and d the private exponent. The public key is the pair (N, e) which is made public and the private key is the pair (N, d) which is kept secret.

RSA encryption and decryption explained in context of the experiment scenario (Section 4.1):

Encryption:

The symmetric key M :
Encrypted key = $M^e \text{ mod } n$

The message digest M :
 Encrypted digest = $M^d \bmod n$

Decrypting:

The symmetric key C :
 Decrypted key = $C^d \bmod n$

The message digest C :
 Decrypted digest = $C^e \bmod n$

Where M is the key or digest converted to an integer according to [RSALab2002], C the encrypted key or digest and n the particular modulus, chosen to be either 512, 1024, 2048, 3072 or 4096.

In particular, it should be noted that encrypting the key and encrypting the message digest is not the same function as one uses the public- and the other the private exponent. Therefore, encrypting the symmetric key and decrypting the message digest (in the verification process) is mathematically equivalent as they both use the public exponent. The same can be said for encrypting the message digest (in the signing process) and decrypting the symmetric key as they both use the private exponent.

RSA operation time greatly depends on the length of e and d [Freeman1999], such that longer exponents incur much larger time overheads. It would therefore be desirable to use smaller values for e and/or d if possible.

4.2.2 Smaller public exponent

We consider how the length of the public exponent affects security as both security mechanisms (Section 4.3) exploit this to achieve faster symmetric key encryption and message verification. The smallest possible value for e is 3 [Boneh1999]. This can however weaken RSA confidentiality assertions. In particular, if $M < \sqrt[e]{N}$ the plaintext can easily be recovered [Rivest2003]. Hastad's broadcast attack can be mounted if k cipher texts, encrypted with the same public exponent, can be collected such that $k \geq e$. [Boneh1999]. The Chinese Remainder Theorem (CRT) can then be used to recover the plaintext message [Eastlake2001, Boneh1999]. A defence against such attacks

would be to 'pad' the message using some random bits [Bellare1994]. Coppersmith imposed further restrictions on this in his "Short Pad Attack" which concludes that for $e = 3$ an attack can still be mounted, even though a random set of bits are used, if the pad length is less than $1/9^{\text{th}}$ of the message length [Boneh1999]. PKCS#1 [Jonsson2003, RSALab2002] does however propose the use of Optimal Asymmetric Encryption Padding (OAEP) [Bellare1994] for new applications and PKCS1-v1_5 for backward compatibility with existing applications.

Although $e = 3$ can provide adequate security, if necessary precautions are taken, the current recommendation is $e = 2^{16} + 1$ [Boneh1999] which is still small, requiring only 17 multiplications, but big enough to solve the above problems at the cost of a slight increase in encryption time. Short public exponents are not however a concern for signature schemes [Eastlake2001, Rivest2003].

4.2.3 Smaller private exponent

A shorter private exponent would result in faster key decryption and message signing. Typically the private exponent is the same length as the modulus regardless of the public exponent length. Wiener [Wiener1990] has however shown that if $d < \frac{1}{3}N^{0.24}$ the private exponent can be obtained from the public key (N, e). Since N is typically 1024 bits long, d must be at least 256 bits long. More recently, Boneh and Durfree have shown this to be closer to $d < N^{0.292}$ [Boneh2000, Sun2005] and predicted the likely final result to be closer to $d < N^{0.5}$ [Boneh1999, Boneh2000].

Other techniques used to decrease algorithm operation time include the use of the Chinese Remainder Theorem [Boneh1999], know as RSA-CRT, which is said to be approximately 4 times faster than using standard RSA algorithms [Sun2005]. Rebalanced RSA-CRT can also be used and tries to shift the cost towards the usage of the public exponent e [Boneh2002, Wiener1990].

4.3 Security software analysis

Java keytool, Java's Key and Certificate Management Tool, is used to create the Java keystore, with appropriate key pairs, used by TSIK and JCE. The keytool generates key pairs where N is user specified (512, 1024 or 2048), d is the same length as N and e defaults to $2^{16} + 1$ (i.e. 17 bits long). As stated in Section 4.2.2 and 4.2.3, these values are adequate and it is currently recommended that the user selects the modulus to be at least 1024 bits.

TSIK 1.10 provides additional functionality, above that of the Java Cryptography Extensions (JCE), to construct valid XML messages after encryption/decryption or signing/verifying. These messages conform to the W3C XML Signature and Encryption specifications [W3C2002]. TSIK supports Triple DES (in cipher block chaining mode) for symmetric encryption, as defined by W3C [W3C2002]. Using a key length of at least 112 bits will currently provide sufficient security. Triple DES is however relatively slow compared to other more recent contenders such as AES [Aslam2004]. Conversely, it has stood the test of time and so is potentially a more reliable solution.

Only SHA-1 is provided for message digest generation (digest length of 160 bits). SHA-1 has very recently been shown to be less secure than predicted and it is recommended that SHA-256 or above should be used [Lenstra2005]. RSAES-PKCS1-v1_5 algorithm, specified by W3C [W3C2002] and [Kaliski1998], is used as the RSA standard. As stated in Section 4.2.2 above; if backward compatibility is not an issue OAEP should be used in preference to PKCS1-v1_5. However, PKCS1-v1_5 provides adequate security assuming the programmer is aware of certain issues. Also, [Kaliski1998] indicates that RSA-CRT is used.

JCE does not support the creation of valid XML messages but supports various symmetric key algorithms including AES, Triple DES and RC5. It also supports SHA-1, SHA-256, SHA-512 and MD5, amongst others, for message digest generation. It also

specifies that the padding is applied according to [RSALab2002]. RSA-CRT is also used.

4.4 Performance analysis

The following section details a comparative evaluation of the performance of VeriSign's TSIK toolkit with respect to the standard Java Cryptography Extensions (JCE) in order to identify whether TSIK is a viable tool to secure time-constrained online transactions.

4.4.1 Environment

All experiments were run on a 3GHz Intel Pentium 4 with 1GB RAM, running Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2-b28) on top of Linux Fedora Core 2. We used *The Legion of the Bouncy Castle* [Legion2005] as the Java RSA provider for both JCE and TSIK, and used Apache Axis 1.2 to generate the appropriate WSDL interface for the web service, which was hosted on Tomcat 5.

Axis was used to both generate the appropriate SOAP messages, from the Java code and TSIK XML documents, to be sent to the web service, also known as the server, and to generate the SOAP messages which are sent back from the web service to the client. We took performance measurements on the client and server side where the TSIK and JCE implementations reside. Message transmission and conversion delays were not measured.

4.4.2 Experiments

We set up three experiments, as detailed below.

Experiment 1:

In experiment 1 we analyse the performance of Triple DES, as function of message size:

- Client side: Message plaintext encrypted using Triple DES with a keysize of 168. Symmetric key encrypted using an RSA public key (Modulus 1024)
- Server side: Encrypted symmetric key decrypted using RSA private key (bit length 1024) and cipher text then decrypted.

Experiment 2:

In experiment 2 we analyse the combined performance of SHA-1 and RSA algorithms, as a function of the message size:

- Client side: Message signed using SHA-1 and RSA private key (bit length 1024)
- Server side: Message verified using SHA-1 and RSA public key

Experiment 3:

In experiment 3 we analyse how the modulus size affects the performance of RSA during signature creation and verification:

- Client side: Message signed (as in experiment 2) using RSA key sizes 512, 1024 and 2048.
- Server side: Message verified.

4.4.3 Results

We executed the above experiments for TSIK as well as JCE. We repeated the first two experiments for messages with a range of plaintext sizes, namely 2, 4, 8, 16, ... , 512 and 1024 kB. Experiment 3 was done using a 2 kB plaintext size. The results are shown in the graphs below. It should be noted that all points in Figures 6 and 8 exhibit confidence intervals of 3 milliseconds and points in Figures 7 and 9 exhibit confidence intervals of 0.1 milliseconds. Both with probability 0.9 (where 1.0 is certain).

Experiment 1:

Figure 6 shows that JCE performs noticeably better for large file sizes. It also shows that Triple DES encryption takes longer than decryption in both cases (TSIK and JCE). Note that the graph also indicates that for very large messages it is decryption that takes longer when using TSIK. We have no precise explanation for this, but suspect it has to do with the particulars of the implementation.

For RSA we see the opposite effect. Figure 7 indicates that RSA encryption takes less time than decryption. As we hinted at earlier in this paper, that is caused by the size of the keys used in encryption and decryption. For encryption, the public key is used, which has a

small public exponent of 17 bits. When comparing TSIK with JCE, we see that the differences are minimal. Decryption varies by an average of about 1 millisecond between the implementations and encryption even less.

Experiment 2:

Figure 8 shows that signing takes more time in both cases. This is once again expected as the messages are signed using the large 1024 bit RSA private key. Encrypting the message digest should take constant time for each file size and so the graph pattern should be wholly due to SHA-1 hashing. Whereas signing and verification time increase steadily for JCE, TSIK performs markedly worse for large file sizes.

Experiment 3:

Figure 9 shows that doubling the RSA key size causes signing time to increase rapidly whilst having little effect on the verification time. This can partly be explained by the fact that doubling the key size effectively doubles the length of the private exponent (used in signing) whilst keeping the public exponent length constant.

4.5 Summary

TSIK is a toolkit to aid secure Web Service interactions. We have shown, through performance measurements, that TSIK has comparable performance to Java's Cryptography Extensions (JCE). Its performance is similar to JCE, except that it slows down when processing messages with large plaintext sizes. It also provides adequate confidentiality, non-repudiation and sender authentication guarantees through the use of Triple DES and RSA, though should consider using SHA-256 for message verification in future releases as is suggested in recent literature [Lenstra2005]. With respect to technical ability, TSIK appears to be a viable and competitive option in securing web based business interactions.

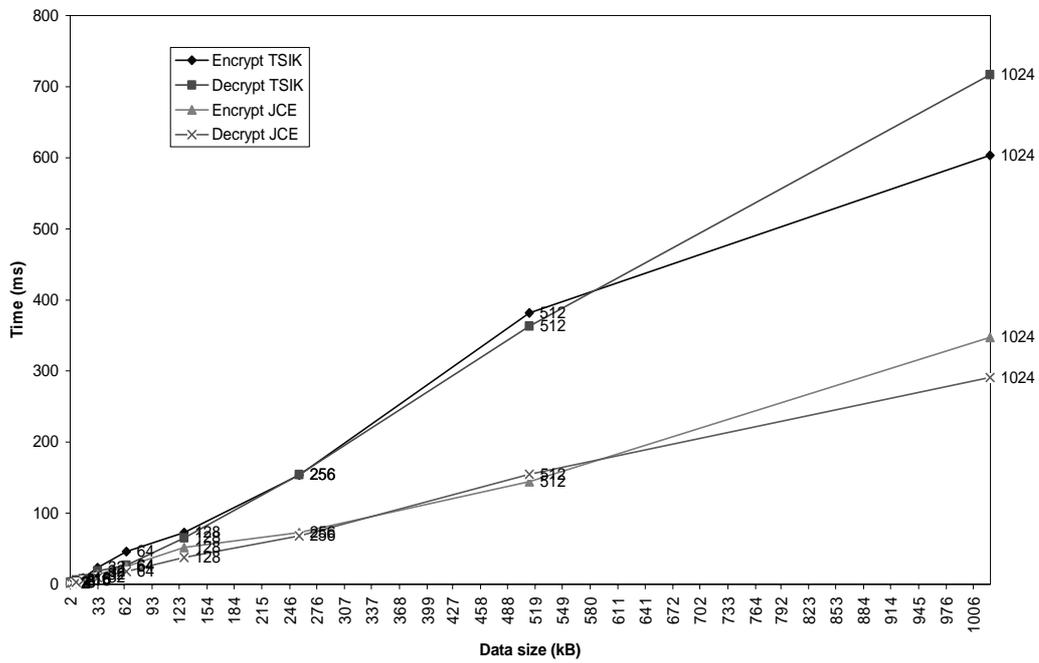


Figure 6: Triple DES encryption time

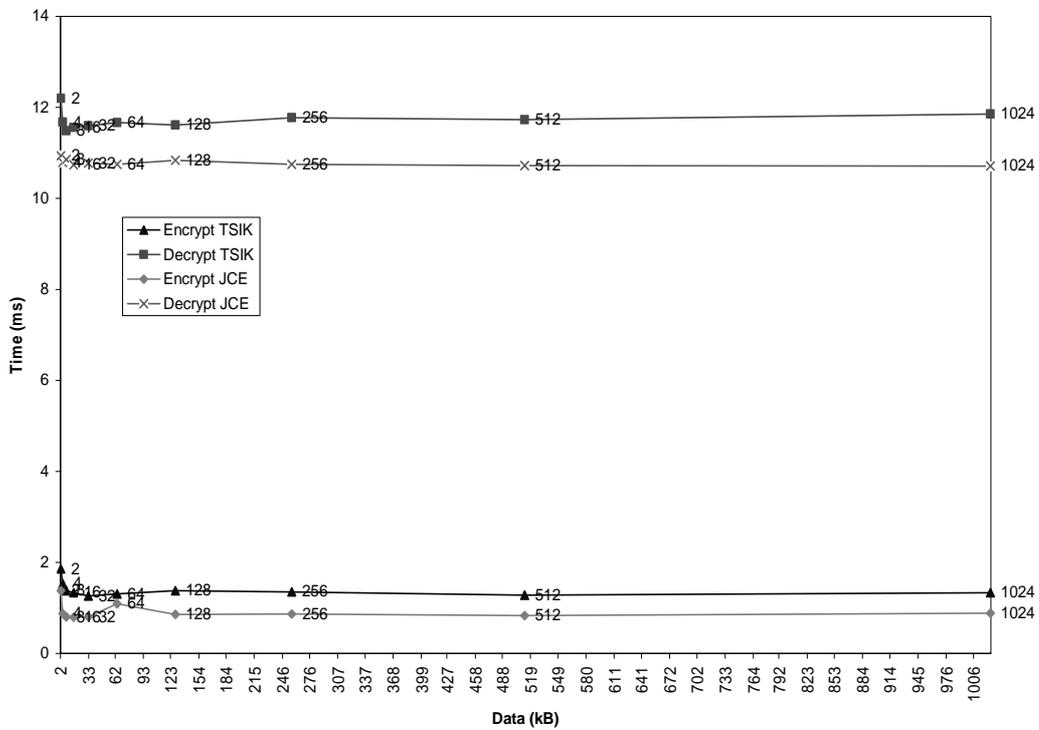


Figure 7: RSA-1024 encryption time of 168 bit Triple DES key

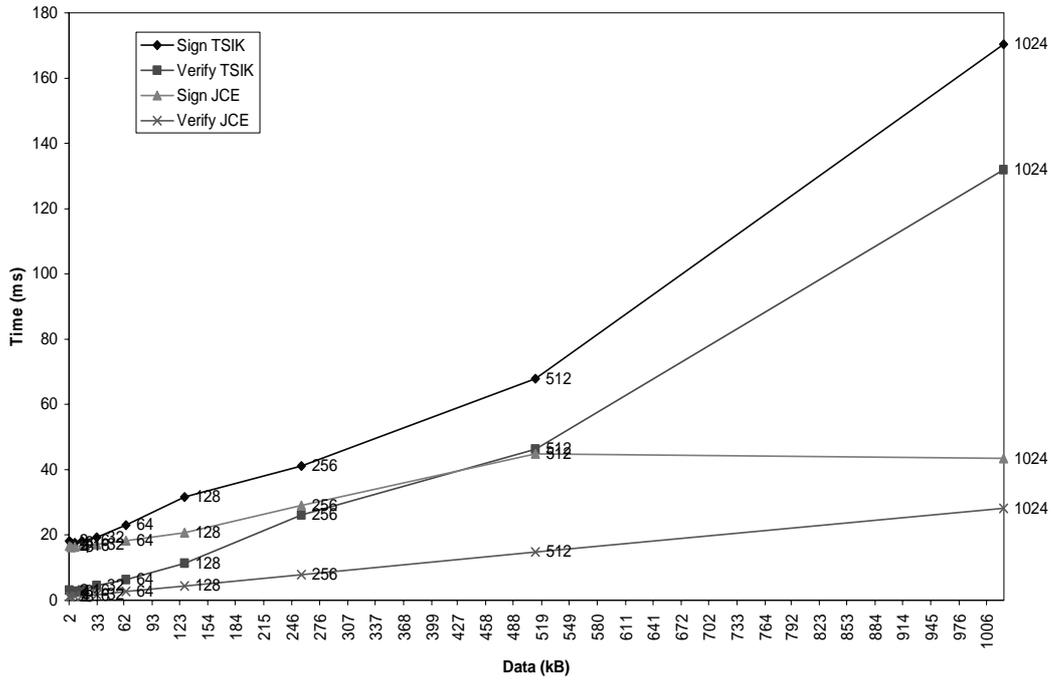


Figure 8: Message signing/verifying (using SHA-1 and RSA-1024)

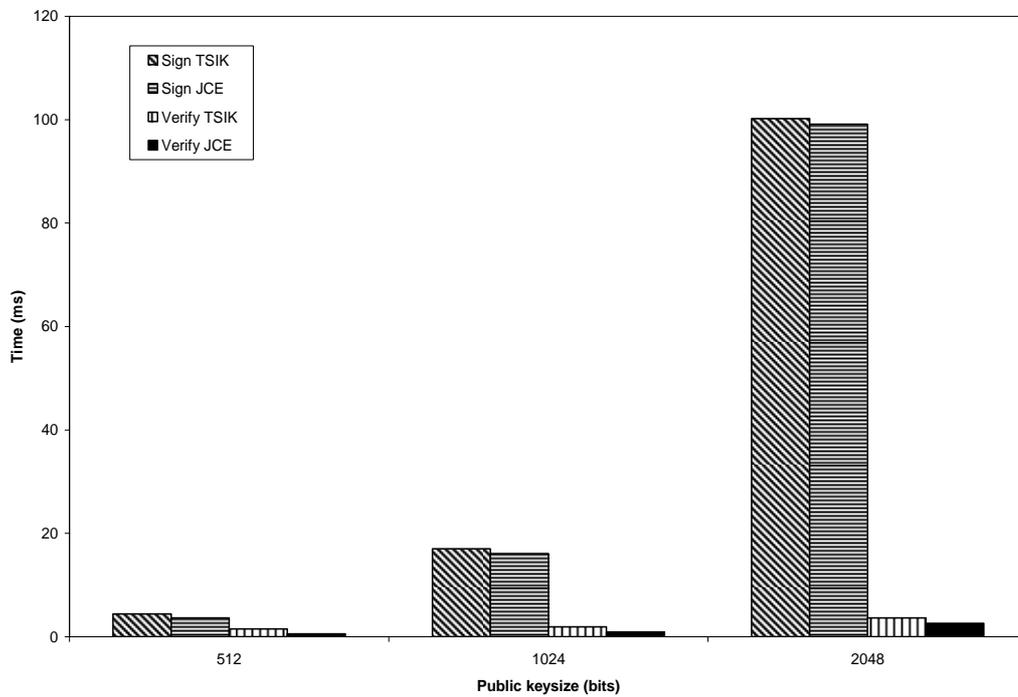


Figure 9: Message signing/verifying (2kB message size)

5. Conclusion

In comparing the performance of encryption algorithms we observed that IDEA was the fastest algorithm available in the distributions we tested. In fact most symmetric algorithms achieved better performance than Triple DES, which was used in our experiments with TSIK and JCE. Furthermore we found that the particular implementation of the algorithm also had a significant impact on the execution time. It therefore seems likely that further improvements can be made to TSIK to make it more suitable for real-time online transactions by selecting algorithms such as IDEA and SHA-256 and also providing fast implementations of those algorithms.

Bibliography

- [Adams2003] C. Adams and S. Lloyd, *Understanding PKI: Concepts, Standards, and Deployment Considerations*, second edition, Addison-Wesley, 2003.
- [Aslam2004] J. Aslam, S. Rafique and S. Tauseef-ur-Rehman, Analysis of Real-time Transport Protocol Security, *Information Technology Journal* 3 (3):311-314, 2004.
- [Bellare1994] M. Bellare and P. Rogaway, Optimal Asymmetric Encryption, In EUROCRYPT '94, *Lecture Notes in Computer Science* volume 950, pages 92-111, Springer-Verlag, 1994.
- [Bloom2002] J. Bloomberg, Testing Web Services Today and Tomorrow, *Rational Edge*, October 2002.
- [Boneh1999] D. Boneh, Twenty Years of Attacks on the RSA Cryptosystem, In *Notices of the American Mathematical Society* (AMS), Vol. 46, No. 2, pp. 203-213, 1999.
- [Boneh2000] D. Boneh and G. Durfee, Cryptanalysis of RSA with Private Key d Less than $N^{0.292}$, *IEEE Transactions on Information Theory*, 46(4):1339-1349, July 2000.
- [Boneh2002] D. Boneh and H. Shacham, Fast Variants of RSA, *CryptoBytes*, 2002, Vol. 5, No. 1, Springer, 2002.
- [Cryptix2005] Cryptix JCE, 2005.
<http://www.cryptix.org/>
- [Eastlake2001] D. Eastlake, RSA/SHA-1 SIGs and RSA KEYS in the Domain Name System (DNS), *IETF Network Working Group*, RFC 3110, May 2001.
- [Freeman1999] W. Freeman and E. Miller, An Experimental Analysis of Cryptographic Overhead in Performance-critical Systems, *MASCOTS*, October 1999.
- [Jonsson2003] J. Jonsson and B. Kaliski, Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography, Specifications Version 2.1, *IETF Network Working Group*, RFC 3447, February 2003.
- [Kaliski1998] B. Kaliski and J. Staddon, PKCS #1: RSA Cryptography Specifications Version 2.0, *IETF Network Working Group*, RFC 2437, October 1998.
- [Legion2005] The Legion of the Bouncy Castle, release 1.30, 2005.
<http://www.bouncycastle.org>
- [Lenstra2005] A. Lenstra, *Further Progress in Hashing Cryptanalysis*, February 26, 2005.
- [Rivest1978] R. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public-key Cryptosystems, *Communications of the ACM*, 21(2):120-126, 1978.
- [Rivest2003] R. Rivest and B. Kaliski, *RSA problem*, To appear in *Encyclopedia of Cryptography and Security* (Kluwer).
- [RSALab2002] RSA Laboratories, *PKCS#1 v2.1: RSA Cryptography Standard*, June 14, 2002.
- [Sun2005] H.-M. Sun and M.-E. Wu, An Approach towards Rebalanced RSA-CRT with Short Public Exponent, *Cryptology ePrint Archive*, 053/2005.
<http://eprint.iacr.org/>
- [SunJCE2005] Sun Microsystems, Inc, Java™ Cryptography Extension, 2005.
<http://java.sun.com/products/jce/index.jsp>
- [Wiener1990] M. Wiener, Cryptanalysis of Short RSA Secret Exponents, *IEEE Transactions on Information Theory*, 36:553-558, May 1990.
- [W3C2002] W3C Recommendation, *XML Encryption Syntax and Processing*, <http://www.w3.org/TR/xmlenc-core>, 10 December 2002.
- [Schneier1996] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*, Wiley, 1996.

The Use of Multi-legged Arguments to Increase Confidence in Safety Claims for Software-based Systems: a Study Based on a BBN Analysis of an Idealised Example¹

Bev Littlewood^{a,*} David Wright^a

^a*CSR, City University, Northampton Square, London, EC1V 0HB*

Abstract

The work described here concerns the use of so-called multi-legged arguments to support dependability claims about software-based systems. The informal justification for the use of multi-legged arguments is similar to that used to support the use of multi-version software in pursuit of high reliability or safety. Just as a diverse, 1-out-of-2 *system* might be expected to be more reliable than each of its two component versions, so a two-legged *argument* might be expected to give greater confidence in the correctness of a dependability claim (e.g. a safety claim) than would either of the argument legs alone.

Our intention here is to treat these argument structures formally, in particular by presenting a formal probabilistic treatment of ‘confidence’, which will be used as a measure of efficacy. This will enable claims for the efficacy of the multi-legged approach to be made quantitatively, answering questions such as ‘How much extra confidence about a system’s safety will I have if I add a verification argument leg to an argument leg based upon statistical testing?’

For this initial study, we concentrate on a simplified and idealized example of a safety system in which interest centres upon a claim about the probability of failure on demand. Our approach is to build a BBN model of a two-legged argument, and manipulate this analytically via parameters that define its node probability tables. The aim here is to obtain greater insight than is afforded by the more usual BBN treatment, which involves merely numerical manipulation.

We show that the addition of a diverse second argument leg can, indeed, increase confidence in a dependability claim: in a reasonably plausible example the doubt in the claim is reduced to one third of the doubt present in the original single leg. However, we also show that there can be some unexpected and counter-intuitive subtleties here; for example an entirely supportive second leg can sometimes undermine an original argument, resulting overall in less confidence than came from this original argument. Our results are neutral on the issue of whether such difficulties will arise in real life - i.e. when real experts judge real systems.

1 Introduction

Assessment of dependability of software-based systems has long been acknowledged to be difficult. There are several reasons for this. Software is often novel, so that claims can rarely be based upon previous experience. Much of the evidence available concerns the software process – how it was built – and not the built product itself. There is a great reliance upon expert judgement – e.g. in how claims about the quality of the build process can be turned into claims about the delivered product’s dependability. There may be doubt about the truth of some of the assumptions that underpin the reasoning used to support a claim, e.g. that a test oracle is correct.

Such uncertainty about dependability claims is particularly important when the systems involved are safety critical. One approach that has been proposed to try to limit and control this uncertainty is the use of multiple diverse arguments to support dependability claims: the idea is analogous to the use of fault tolerance to make systems reliable. Thus each of the diverse arguments could, in principle, support the claim but might be undermined by doubt about underlying assumptions, weakness of evidence, etc.

In recent years, some standards and codes of practice have suggested the use of diverse arguments. In UK Def Stan 00-55 [1], for example, it was suggested that one leg be based upon logical proof of correctness, the other upon statistical testing. Argument legs are sometimes quite asymmetric: for example, in [2] the first leg is potentially complex, whereas the second leg is deliberately simple. Occasionally, the only difference between the legs lies in the people involved, e.g. in independent verification and validation. This last case can be plausible when there is a paucity of hard empirical evidence upon which to base the arguments, and thus necessarily a large element of expert judgement is used – different teams might provide some protection against identical human mistakes.

The differences shown in these examples reflect, we believe, the need for better understanding about the use of diversity in arguments. At an informal level, diversity seems plausibly to be ‘a good thing’, just as it is for achieving system dependability, but there is no theoretical underpinning to such an assertion. For example, we do not know what are the ‘best’ ways to use diversity (nor even exactly what ‘best’ means here); we do not know how much we can claim for the use of diversity in a particular case.

* Corresponding author.

¹ **This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible.**

Our aim in this paper is to provide the beginnings of a formalism to answer some of these questions, and provide support for the ‘diversity approach’. We shall look at the combination of multiple argument legs in a part of a safety case for a critical system. The difficult questions here concern how to combine the disparate evidence and assumptions that form the different legs. There are several such questions that might be of interest. For example, we might want to know whether multi-legged arguments are efficient in the sense of being cost-effective: e.g., for a given outlay, would it be better to divide this between a proof and a testing leg, or to spend it all on a larger test?

At an informal level, one can take an argument leg to comprise: some *assumptions*, some *evidence*, and some *reasoning* that allow a dependability *claim* to be made at a certain level of *confidence*. Typically, such an argument leg will support an infinite number of different (claim, confidence) pairs – the more stringent the claim, the lower the confidence that will come from a particular argument leg. We shall interpret ‘confidence’ to be a probability (that the claim is true). This probability, in turn, will be interpreted as the usual Bayesian subjective strength of belief in the claim, held by an individual whom we shall refer to as ‘the expert’. This person might be a regulator, or some other person who has to take a decision on the acceptability of the system.

It is thus confidence that allows us to discuss the ‘strength’ of arguments: for example, an argument that allows someone to place 99% confidence in a claim that a system’s probability of failure on demand is less than 10^{-3} is clearly ‘stronger’ than an argument that only allows them to place 90% confidence in the same claim. For simplicity, in this paper we shall always consider the claim to be fixed – perhaps arising from some wider safety case – and thus compare arguments solely via the confidence they engender in this claim. Clearly this is not the only way one could proceed, but it will suffice for our purposes here.

It is easy to see that confidence – and its complement, ‘doubt’ – will depend upon: confidence/doubt in the truth of the assumptions underpinning the argument; strength and/or extensiveness of the evidence; correctness of the reasoning. Continuing in this informal vein, it seems plausible that for *multi-legged* arguments the overall effectiveness will depend upon the same factors, and in addition the *dependence* between the legs. Thus we might expect a two-legged argument whose legs are ‘very diverse’ to be more effective – i.e. give greater confidence – than one where the legs are very similar (all things being equal).

Of course, as we have been at pains to state, all this is very informal. Our intention in the work reported here is to put these ideas onto a formal basis.

We shall use as the basis of the paper a more formal treatment of an example first examined in [3, Example 1, p27]. In this example a two-legged argument

was proposed. Firstly, a leg based upon statistical evidence from operational testing and the use of an oracle produces a claim for a particular probability of failure upon demand (pfd). It is reasoned that this pfd represents a sufficiently small risk during the expected operational life of the system. To this part of the argument is added a second leg based instead on logical reasoning which is assumed to produce a claim for complete perfection of operational behaviour (at least with respect to a subclass of failures). Here, the second leg produces a claim of complete freedom from (a class of) faults. If the overall argument is intended to support a claim of (better than) 10^{-3} pfd, then only the statistical *testing leg* addresses this directly. Nevertheless, it is easy to see how the logical leg can provide additional support: if the statistical evidence alone gives 99% confidence that the pfd is smaller than 10^{-3} then the additional *verification leg* might allow this level of confidence in 10^{-3} to be increased.

Note that for these individual legs important sources of doubt in the claim are doubt about the correctness of the oracle (for the testing leg), and doubt about the correctness of the specification (for the verification leg). Furthermore, such doubts are likely to be dependent: in particular, doubt in the correctness of the specification is likely to affect doubt in the correctness of the oracle. We might expect that the greater the doubt in the specification, the greater the doubt in the oracle. Clearly, these doubts will propagate and affect the confidence associated with the use of both arguments in a two-legged configuration: we might expect this confidence to be less than would be the case if the specification and oracle doubts were independent.

Note also that dependence between legs in this example can arise in other ways. It could arise from the *evidence*, for example: the observation of a failure in the testing leg would completely refute the perfection claim of the second leg.

Issues of (lack of) dependence here are very similar to those that arise in *system* diversity, where it has been established from theoretical [4,5] and empirical [6] studies that independence is extremely elusive. If this is also true of arguments we would need to be sceptical of simplistic claims, e.g. that 99% confidence in a claim could be justified from a two-legged argument based on two legs each of which alone only allow 90% confidence. Proper understanding of argument *dependence* is therefore an important goal of this research. It turns out, in fact, that issues of dependence between legs can be subtle and counter-intuitive.

We realise that much of what we say here applies to dependability assessment of systems in general, but the issues discussed are often particularly acute for software based systems, where there may be great complexity and novelty in the system design, and there is typically a large reliance in the safety assessment on expert judgement.

In this paper we mainly concentrate on the problems associated with the assessment of confidence. We only address issues of decision-making – e.g. whether to accept and deploy a system – briefly, and then only to treat the case of ‘dangerous’ argument failure, i.e. the acceptance of an untrue claim. The other kind of failure - rejecting a claim when it is true - will also be important (e.g. it may have important economic consequences), but will not be considered here.

The paper is organised as follows. We begin by describing a 6-variable Bayesian belief network (BBN) representing the structure of the two-legged argument example. A BBN topology for this system assessment is first presented, with an enumeration of the *independency model*² which it represents.

There follow proposals for the content of those parts of the node probability tables that would be required in order to deal with the observation case which is of greatest practical importance for the application. This is the ‘complete success’ case, which we shall term the *ideal observation* case for this two-legged argument. For this, we suppose that the execution testing discovers *no failures at all* (*testing leg* success); and furthermore, the system is formally verified correct against its specification (*verification leg* success). This case is of practical importance in some safety-critical industries, where it is the *only* case which would allow acceptance of a system for operational use. In the UK nuclear industry, for example, failures in test would be unacceptable regardless of what could be inferred statistically from the test result.

Our allocation of node probability tables used to analyse this ideal observations case is parameterized, i.e. it specifies the conditional probabilities of each node, given its parents’ values, numerically except for unspecified values of a set of independent model parameters. There are 12 independent model parameters significant for the analysis of the ideal observation case. A particular expert’s beliefs will thus be represented by an assignment of numerical values to these parameters.

A substantial part of the remainder of the paper examines the consequences of different expert beliefs, concentrating on questions of when the two-legged approach is effective, and what are the factors that determine its effectiveness. We show that there are some unexpected subtleties here, and give examples of some surprising and non-intuitive results.

² Other synonymous terms and some references are given on p7.

2 Model Variable Definitions and BBN Topology

The construction of our model proceeds in the usual stages of: model variable identification, definition and respective state-space construction; BBN topology construction, to represent graphically assumed conditional independencies (CIs) among the model variables; and finally local node conditional probability table definition. In this ordered presentation of the process, the discoveries and difficulties encountered during later stages may well feed back into adjustments and refinements to earlier stages.

The first stage in building a BBN is the identification of model variables. Our model variables are defined in the following list, which gives in square brackets the state-space we have used, in this paper, for each variable. For the sake of simplicity in this initial model formulation, the state-spaces are all Boolean, apart from the first. In some cases, this is a deliberate simplification of the real situation which we do not intend to retain in all our future work.

- S - The system's unknown, true probability of failure on demand (pfd) [$0 \leq S \leq 1$].
- Z - system *specification* [$Z \in \{correct, incorrect\}$]. A system verification is performed directly against this specification, to form one leg of the system dependability argument.
- V - conclusion from the *verification* of the system against its specification [$V \in \{verified, not\ verified\}$]. For the purposes of the current investigation we shall be interested only in the *ideal observation* outcome that the system is verified correct.
- O - *oracle* used in system testing (by execution of the system) [$O \in \{correct, incorrect\}$]. In this current BBN, we have for simplicity made the unrealistic assumption that the *operational profile*, used to simulate the test inputs, is perfectly representative of the statistical pattern occurring during real use.
- T - system *test results* [$T \in \{no\ failures, failures\}$]. We shall be interested here only in the *ideal observation* case of *no failures*.
- C - acceptance (or otherwise) of final *claim* as to whether or not the system is fit for use [$C \in \{accepted, rejected\}$]. Of course, C 's two parents, T and V , are both *observable* nodes. So in one usage scenario we have in mind of this model, C itself will be a 'deterministic' node, in the sense that its realised value will be a chosen deterministic function of its parents' values. For the purposes of illustration throughout this paper we will use the claim that the operational system pfd S is better than 10^{-3} .

The ultimate purpose of the following BBN model is to derive posterior distributions of random variables, S, C , of *practical importance* (goal variables), following observation of other variables T, V whose values are *directly measurable* (or amenable to direct human assessment). The model, like most other

BBN models, also includes essential model-structural ‘mediating variables’, Z, O , in this case, which fall into neither of these two categories. Formally, the BBN topology encodes a system of conditional independence assumptions, collectively termed a *Markov model*, *(in)dependency relation*, *(in)dependency model*, or *Conditional Independence (CI) relation* – see p91 of [7], §2.4 of [8], and p5 of [9] – which enable the grand multivariate distribution of all model variables to be composed of several node conditional probability distributions of lower dimension. Consequently the required posterior distributions, of goal variables given evidence, can likewise be expressed, in §3 below, in terms of these node conditional distributions. The CI assumptions encoded in the topology justify the derivations involved, and the pictorial representation of this topology, properly understood, efficiently communicates these CI assumptions (some more evidently than others). Thus probabilistic CI assumptions and the BBN topologies that encode them are devices for constructing, communicating, and reasoning with multi-variate probability models.

The theoretical study of how a probabilistic CI model may be precisely represented by a graph is based on an analogy of probabilistic CI relations with various notions of the separation of two sets of graph nodes by a third ‘separator’ set of nodes. The formal notions of ‘d-separation’, ‘graphoid’, ‘semi-graphoid’, and ‘I-map’ are central to this theory. See [7–10] for further details.

Our BBN topology is shown in Figure 1.

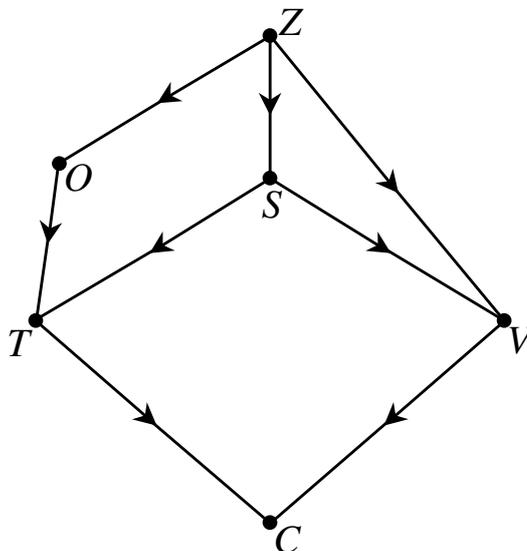


Fig. 1. BBN model topology

Expressed algebraically, the ‘CI statement’ (or just ‘CI’, for short), “ \mathcal{A} is conditionally independent of \mathcal{B} , given \mathcal{S} ”, can be thought of as a factorization property of the joint distribution function:

$$\mathcal{A} \perp\!\!\!\perp \mathcal{B} | \mathcal{S} \quad \text{means} \quad P(\mathcal{A}, \mathcal{B}, \mathcal{S}) = P(\mathcal{A} | \mathcal{S}) P(\mathcal{B} | \mathcal{S}) P(\mathcal{S}), \quad (1)$$

where $\mathcal{A}, \mathcal{B}, \mathcal{S}$, here represent *sets* of model *random variables*³. Thus, each CI assumption asserts that joint (or “joint conditional”) probability distributions will *factorize* – in precisely stated ways – into products of other *conditional* joint distributions, *each involving fewer variables*⁴. Our Markov model was constructed in the BBN form of Figure 1 by explicitly making the CI assumptions that each graph node is conditionally independent, given its parents, of its other non-descendants. Other CI statements are logical consequences of these. (See e.g. [7,11] for precise definitions and theory of how to use the net topology to determine all its logical CI consequences.) The term *conditional dependence* denotes simply the absence of a specified CI factorization.

The graph topology of our BBN can be thought of as an embodiment of a Markov model, i.e. a complete and consistent⁵ set of CI beliefs of an expert concerning the model random variables. There are many ways of specifying the dependency model which this graph topology represents. E.g. one economical, logically independent set of CI assumptions which together completely specify the model is:

$$\begin{aligned} O \perp\!\!\!\perp SV \mid Z \\ T \perp\!\!\!\perp ZV \mid OS \\ C \perp\!\!\!\perp OSZ \mid VT \end{aligned}$$

Expressed in terms of its 26 *elementary CI statements*⁶ as in [8], the same dependency model can be written

$$\begin{array}{cccccc} O \perp\!\!\!\perp S \mid Z & O \perp\!\!\!\perp S \mid ZV & & & & \\ O \perp\!\!\!\perp V \mid Z & O \perp\!\!\!\perp V \mid ZS & O \perp\!\!\!\perp V \mid SZT & O \perp\!\!\!\perp V \mid SZTC & & \\ Z \perp\!\!\!\perp T \mid OS & Z \perp\!\!\!\perp T \mid OSV & Z \perp\!\!\!\perp T \mid OSVC & & & \\ V \perp\!\!\!\perp T \mid OS & V \perp\!\!\!\perp T \mid OSZ & V \perp\!\!\!\perp T \mid SZ & & & \\ Z \perp\!\!\!\perp C \mid VT & Z \perp\!\!\!\perp C \mid VTO & Z \perp\!\!\!\perp C \mid VTS & Z \perp\!\!\!\perp C \mid VTOS & Z \perp\!\!\!\perp C \mid OSV & \\ O \perp\!\!\!\perp C \mid VT & O \perp\!\!\!\perp C \mid VTS & O \perp\!\!\!\perp C \mid VTZ & O \perp\!\!\!\perp C \mid VTSZ & O \perp\!\!\!\perp C \mid SZT & \\ S \perp\!\!\!\perp C \mid VT & S \perp\!\!\!\perp C \mid VTO & S \perp\!\!\!\perp C \mid VTZ & S \perp\!\!\!\perp C \mid VTOZ & & \end{array}$$

³ Other forms, such as $P(\mathcal{A}, \mathcal{B} \mid \mathcal{S}) = P(\mathcal{A} \mid \mathcal{S})P(\mathcal{B} \mid \mathcal{S})$, are for most practical purposes equivalent; although there may be occasional exceptions relating to conditioning on zero-probability \mathcal{S} -events. Some authors use the very inclusive definition, avoiding potential zero divisions, $P(\mathcal{A}, \mathcal{B}, \mathcal{S})P(\mathcal{S}) = P(\mathcal{A}, \mathcal{S})P(\mathcal{B}, \mathcal{S})$.

⁴ *Conditional* independence means distinct factors may contain common variables.

⁵ No set of CI assertions can in itself exhibit logical inconsistency. The associated Markov model includes all the logical CI consequences of the explicitly asserted CI statements. We need all of these to be consistent with any conditional *dependence* beliefs expressed by the same expert.

⁶ [8] shows that *every* probabilistic dependency model is completely characterised by listing its elementary CI statements.

When an expert declares himself satisfied with a BBN topology such as that in Figure 1, he is really saying that he believes the CI assertions that are entailed by the topology. Thus, part of the topology elicitation exercise would be an exposure to, and acceptance of, these.

The idea captured by the ternary relation $\mathcal{A} \perp\!\!\!\perp \mathcal{B} | \mathcal{S}$ can be expressed less formally by the statement: ‘*Observation of \mathcal{S} renders \mathcal{A} irrelevant to \mathcal{B}* ’, [7]. A probabilistic uncertainty model implies an $\mathcal{A} \leftrightarrow \mathcal{B}$ -symmetry of this statement. Care is required with its interpretation: The term “observation” denotes *complete* observation, in the sense that the values of all variables in \mathcal{S} should be made *exactly* known before a person interested (solely) in the values of \mathcal{B} will lose interest in information about variables \mathcal{A} . See [7,11–16], and §1.2.1 of [9]. For example, our model assumes $V \perp\!\!\!\perp T | SZ$. A factorization such as

$$P(VT | S > 10^{-3}, Z = \text{correct}) = P(V | S > 10^{-3}, Z = \text{correct})P(T | S > 10^{-3}, Z = \text{correct})$$

does *not* follow; whereas

$$P(VT | S = 10^{-3}, Z = \text{correct}) = P(V | S = 10^{-3}, Z = \text{correct})P(T | S = 10^{-3}, Z = \text{correct})$$

is logically entailed within our model.

This particular CI assumption also illustrates another important point about the above informal interpretation of CI assumptions. In practice we may still incorporate in our model such a CI assumption, even though we may not expect to observe variable(s) \mathcal{S} , or where \mathcal{S} may be in principle impossible to observe. Then it is *hypothetical* exact knowledge of \mathcal{S} that is the conceptual device used to interpret the above CI assumption. In practical model building, many CI assumptions may be of this form, in which the precise values of conditioning variables of some CI assumptions are never expected to be known, as with the assumption $V \perp\!\!\!\perp T | SZ$ in our model.

To state one last clarification about the interpretation of CI assumption $\mathcal{A} \perp\!\!\!\perp \mathcal{B} | \mathcal{S}$, note that the conditioning knowledge-state, assumed to produce the irrelevance of \mathcal{A} to \mathcal{B} , consists of knowing *only* the exact value of \mathcal{S} . The irrelevance can later be destroyed by subsequently acquired knowledge, exact or approximate, of other variables. So strictly, assumption $\mathcal{A} \perp\!\!\!\perp \mathcal{B} | \mathcal{S}$ says that when \mathcal{S} is exactly known, \mathcal{A} is irrelevant to \mathcal{B} for only just so long as the state of all other model variables remains completely unknown, i.e. while we do not discover *anything else than* the value of \mathcal{S} .

Figures 2 and 3 show the BBNs representing an obvious pair of single-legged arguments whose ‘combination’, in some sense, we have discussed up to now, in the form of the BBN model shown in Figure 1:

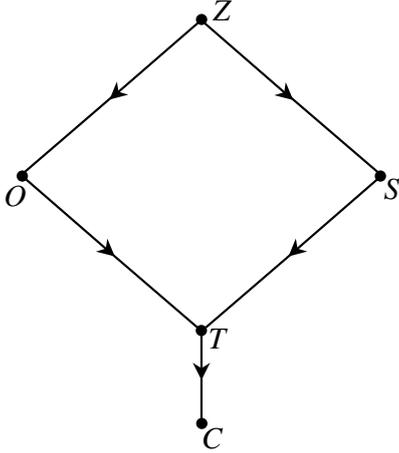


Fig. 2. Testing leg BBN topology

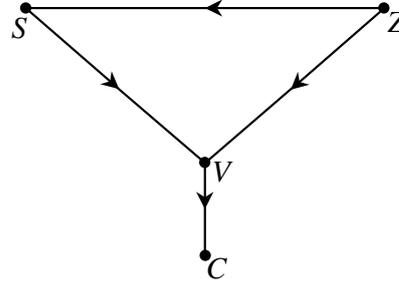


Fig. 3. Verification leg BBN topology

These topologies may be obtained from that of Figure 1 by removing one or other of the two observable nodes V and T . It may be that an expert who accepts the dependency model of Figure 1 as correct for the uncertainties inherent in the two-legged argument would feel the same about Figures 2 or 3 for an argument in which only one of these two source of evidence is available. More rigorously, the relationship between these one-legged and two-legged argument topologies is not trivial. We do not envisage that the appropriate one-legged dependency model should necessarily be merely a ‘marginal’ of the two legged model, obtained by summation over the observable V or T that is to be removed. For example, in all the example uses of these topologies which follow, we choose to assign the values of C , “claim acceptance”, deterministically in terms of C ’s (one or two) parents’ values. This means that our single-legged models will attribute to variable C a different stochastic relationship with the remaining model variables. (Below, in our model based on the topology of Figure 2, we make C a deterministic function of T alone: in our two-legged model based on Figure 1 we do not.) In what follows when we have assigned node conditional probabilities to our topologies, we will say a little more about the relationship between the three multivariate probability models that result.

Represented algebraically, the two single-leg dependency models are

Testing Leg	Verification Leg
$O \perp\!\!\!\perp S \mid Z$	$C \perp\!\!\!\perp SZ \mid V$
$T \perp\!\!\!\perp Z \mid OS$	
$C \perp\!\!\!\perp OSZ \mid T$	

the first OS -symmetric; and the second SZ -symmetric⁷. In the form of ex-

⁷ The latter symmetry is not inherited by Figure 3 itself, though there are derivable

haustive lists of elementary CIs, we have:

<u>Testing Leg</u>				<u>Verification Leg</u>	
$O \perp S Z$					
$Z \perp T OS$	$Z \perp T OSC$				
$Z \perp C T$	$Z \perp C TO$	$Z \perp C TS$		$Z \perp C V$	$Z \perp C VS$
$Z \perp C OS$	$Z \perp C TOS$				
$O \perp C T$	$O \perp C TS$	$O \perp C TZ$	$O \perp C TSZ$		
$S \perp C T$	$S \perp C TO$	$S \perp C TZ$	$S \perp C TOZ$	$S \perp C V$	$S \perp C VZ$

Readers may have noticed that, although we have informally treated a two-legged argument as a combination of single legs, our more formal treatment above has moved in the reverse direction. Comparison of these last CI lists with the one on p8 clearly does not indicate any formal operation allowing composition of these two single-legged dependency models to create the two-legged model. The two-legged model of Figure 1 is a construction embodying various subjective beliefs about the dependencies arising in practice among the variables of our particular application. In particular, our two-legged dependency model can be characterised neither as weaker nor as stronger than the simple conjunction⁸ of the two separate single-legged dependency models.

Precisely, using the elementary CI statement dependency model characterization, the two-legged model deletes all CIs from one single-legged model, most CIs from the other, and of course introduces several other CIs whose contexts⁹

canonical graph representations which *always* show the same symmetries as the dependency model they represent, such as the “largest chain graph” [8,17] model representation.

⁸ i.e. the pooled set of conditional independencies involving the 6 variables

⁹ the *context* of a CI statement is the set of all the model variables it involves: $\text{context}(A \perp B | C) = A \cup B \cup C$.

are not contained in the variable set for either single leg.

<u>Deleted from</u> <u>Testing Leg</u>	<u>Deleted from</u> <u>Verification Leg</u>	<u>Appended</u>
		$O \perp S ZV$
		$O \perp V Z \quad O \perp V ZS$
		$O \perp V SZT \quad O \perp V SZTC$
$Z \perp T OSC$		$Z \perp T OSV \quad Z \perp T OSVC$
		$V \perp T OS \quad V \perp T OSZ$
		$V \perp T SZ$
$Z \perp C T \quad Z \perp C TO$	$Z \perp C V$	$Z \perp C OSV \quad Z \perp C VT$
$Z \perp C TS \quad Z \perp C TOS$	$Z \perp C VS$	$Z \perp C VTO \quad Z \perp C VTS$
$Z \perp C OS$		$Z \perp C VTOS$
$O \perp C T \quad O \perp C TS$		$O \perp C VT \quad O \perp C VTS$
$O \perp C TZ$		$O \perp C VTZ \quad O \perp C VTSZ$
$S \perp C T \quad S \perp C TO$	$S \perp C V$	$S \perp C VT \quad S \perp C VTO$
$S \perp C TZ \quad S \perp C TOZ$	$S \perp C VZ$	$S \perp C VTZ \quad S \perp C VTOZ$

Clearly many of these changes involve variable C and relate to its changed role, mentioned above, in the dependence structure as we move between these three dependency models.

3 Computations from these BBN Topologies

We are primarily interested in the updated joint probability distribution $P(CS | \text{observations})$, particularly the value $P(C = \text{accepted}, S > 10^{-3} | \text{observations})$ concerning an unsafe failure of the entire, two-legged assessment activity. Starting from this BBN model, the observations available will typically consist of values for the pair V, T of model variables. Under the conditional independence assumptions comprising this dependency model, the joint distribution of these four variables has a representation

$$P(CSVT) = P(C|VT) \left\{ \sum_Z P(V|SZ)P(S|Z) \sum_O P(T|OS)P(OZ) \right\} \quad (2)$$

For any pair of observed values (V, T) , we obtain the desired, updated distribution for (C, S) by normalising

$$\begin{aligned}
P(CS|VT) &= \frac{P(C|VT) \left\{ \sum_Z P(V|SZ)P(S|Z) \sum_O P(T|OS)P(OZ) \right\}}{\sum_C \int_S P(C|VT) \left\{ \sum_Z P(V|SZ)P(S|Z) \sum_O P(T|OS)P(OZ) \right\} dS} \\
&= \frac{P(C|VT) \left\{ \sum_Z P(V|SZ)P(S|Z) \sum_O P(T|OS)P(OZ) \right\}}{\int_S \left\{ \sum_Z P(V|SZ)P(S|Z) \sum_O P(T|OS)P(OZ) \right\} dS} \quad (3)
\end{aligned}$$

We have used integration over our single continuous model variable S here, interpreting $P(S|Z)$ as a density function. In fact, notice that, since we are to accept that perfection ($S=0$) is possible, then we must allow mixed distributions for S (these often involving other model variables too, being joint or conditional distributions). In this notation, this means thinking of integrands over S as potentially exhibiting ‘delta-function-like’ behaviour, at $S=0$. (One could use Lebesgue-Stieltjes integrals [18] to notate this more rigorously.)

4 Node Probability Assumptions

4.1 Simplifying and Conservative Assumptions

We start with some assumptions that will simplify the mathematics. Some of these assumptions are quite strong. Below in §4.2 we consider some specific parametric refinements of some of these starting assumptions.

• **Determinism of Claim Acceptance** An assumption, for our *ideal observations* case of success on both argument legs, $(V, T) = (\textit{verified}, \textit{no failures})$, which simplifies the above considerably, is the conditional probability table entry

$$P(C=\textit{accepted} \mid (V, T)=(\textit{verified}, \textit{no failures})) = 1. \quad (4)$$

That is to say, the value of the claim C is fully determined by these observed values of (V, T) alone, so that $P(C=\textit{accepted}, S \mid (V, T)=(\textit{verified}, \textit{no failures})) = P(S \mid (V, T)=(\textit{verified}, \textit{no failures}))$. For this observed (V, T) , (2) and (3) then both become zero for $C=\textit{rejected}$, while, for $C=\textit{accepted}$, (2) loses the term to the left of the braces, as does the numerator of (3). Thus, substituting

into (3) leaves us with the expression

$$P\left((C, S)=(\text{accepted}, s) \mid \text{ideal obs.}\right) = P\left(S=s \mid \text{ideal obs.}\right) = \frac{\sum_Z P(V=\text{verified} \mid S=s, Z)P(S=s \mid Z) \sum_O P(T=\text{no failures} \mid O, S=s)P(OZ)}{\int_S \left\{ \sum_Z P(V=\text{verified} \mid SZ)P(S \mid Z) \sum_O P(T=\text{no failures} \mid OS)P(OZ) \right\} dS} \quad (5)$$

for the updated probability density of S , given the *ideal observations* (for both argument legs). Here, the condition “ \mid *ideal obs.*” is a shorthand for “ $\mid (V, T)=(\text{verified}, \text{no failures})$ ”. Keep in mind that we can interpret the numerator and denominator of the conditional probability (5) as, respectively, an unconditional probability density, and an unconditional probability, in the usual way. See e.g. (13) on p17 below.

It is the behaviour of this formula (5) for the distribution of the pfd S conditionally given the ideal observations that forms the focus of the remainder of the paper.

• **Verification Fallibility Against Correct Specification** Against a correct specification, an *infallible* verification procedure would pass the system precisely if its true pfd is zero: Provided the specification is correct, any positive pfd, however small, will always have been caused by a fault, which will certainly show up as a failure to verify the system. Conversely, one might assume that all systems which fail the verification have non-zero true pdfs. Instead we introduce a pair of *verification fallibility* parameters α, ξ allowing the breakdown of both of these ideal behaviours of the verification process:

$$P(V=\text{verified} \mid S=s, Z=\text{correct}) = \begin{cases} \xi & \text{if } 0 < s \leq 1, \text{ or} \\ 1 - \alpha & \text{if } s = 0, \end{cases} \quad (6)$$

Thus, our model allows that, against a correct specification:

- a perfectly reliable system *can* fail¹⁰ the verification, with probability α ;
- a system having a positive pfd *can* pass the verification, with probability ξ .

For simplicity, we assume that the probability (that is, conditionally given $\langle S=s, Z=\text{correct} \rangle$) of the latter kind of verification failure is independent of the actual (positive) value s of variable S . (Of course this constraint could be relaxed in future models, perhaps by using a parametric function $\xi(s)$.) Note

¹⁰ We have to be careful about terminology here: Surely there may be systems which contain ‘faults’ while being perfectly reliable. (E.g. defective functionality may not be exercised by a particular operational profile; or the system may contain internal fault-tolerance which eliminates the possibility that a certain ‘fault’ could ever result in a system failure.)

that the special case $(\alpha, \xi) = (0, 0)$ restores the *infallibility* assumption for the verification process (provided the specification is correct), as outline above.

- **Conservative Assumption for Incorrect Specification:** Against an incorrect specification, we make the conservative (i.e. pessimistic, taking the perspective of the overriding undesirability of accepting a bad system) assumption that any system will always *pass* a verification against this specification.

$$P(V=verified | S=s, Z=incorrect) = 1. \quad (7)$$

- **Geometric Time-to-Failure Distribution** during testing. We assume that n test inputs cause failures independently with the operational pfd S , which are detected with certainty (and with no false alarms) if the oracle is *correct*. So the probability table of variable T has

$$P(T=no failures | S=s, O=correct) = (1 - s)^n. \quad (8)$$

- **Conservative Assumption for Incorrect Oracle:** To address the case of an incorrect oracle, we again adopt a conservative assumption, similar to that used above for the case of verification against an incorrect specification:

$$P(T=no failures | S=s, O=incorrect) = 1. \quad (9)$$

- **Stochastic Ordering Constraint** We propose for the conditional distribution of S given Z a requirement for the following kind of stochastic ordering as a function of Z

$$P(S > s | Z=correct) < P(S > s | Z=incorrect), \quad \text{for all } 0 \leq s < 1. \quad (10)$$

where it is allowed that either or both of these distributions can have mass concentrated at $s=0$, subject to this inequality.

4.2 Distributional Assumptions

We begin by introducing some shorter notation for those probabilities which will not now be substituted by a parametric distribution:

We will use the symbol π for the unconditional joint distribution of variables ZO , taken in that order, with a first index to represent the Z value, and a second index to represent O . That is, we name four unconditional probabilities,

$\pi_{cc} + \pi_{ci} + \pi_{ic} + \pi_{ii} = 1$, with c for correct, i incorrect. We also use a “wildcard notation”, $*$, for the marginal distributions of Z and of O so, for example, $\pi_{c*} = \pi_{cc} + \pi_{ci} = P(Z=correct)$. Later, we will sometimes display these prior probabilities of variables ZO using a 2×2 matrix layout

$$\begin{array}{cc}
 & O \\
 & \text{correct} \quad \text{incorrect} \\
 Z \text{ correct} & \pi_{cc} \quad \pi_{ci} \quad | \quad \pi_{c*} \\
 \text{incorrect} & \pi_{ic} \quad \pi_{ii} \quad | \quad \pi_{i*} \\
 \hline
 & \pi_{*c} \quad \pi_{*i} \quad | \quad 1
 \end{array} \tag{11}$$

This matrix represents an important set of prior beliefs, since it is here that is captured the dependence between our doubts about specification and oracle correctness. There is likely to be positive “assumption dependence” here, which will presumably cause dependence between the argument legs and undermine, to some extent, the efficacy of the two-legged approach.

For the conditional distribution $P(S|Z)$, we have the complication that it may be a mixed distribution. In our parametric examples we assume this to be continuous on $S \in [0, 1]$ except for a possible concentrated mass at $S=0$. Denote the two concentrated masses p_{0c} and p_{0i} , where the c or i indicates the conditioning value of Z . (We will require $p_{0c} > p_{0i}$ in compliance with assumption (10).) For the sake of statistical conjugacy [19, Ch. 9] with the discrete time model (8), we will use a beta distribution for the continuous component. So for $0 < s \leq 1$, use

$$\text{pdf}(s|Z) = \begin{cases} \frac{(1-p_{0i})}{\beta(a,b)} s^{a-1} (1-s)^{b-1}, & \text{if } Z=incorrect, \text{ or} \\ \frac{(1-p_{0c})}{\beta(a',b')} s^{a'-1} (1-s)^{b'-1}, & \text{if } Z=correct, \end{cases} \tag{12}$$

for some $a, b, a', b' > 0$. Note that if $\xi=0$, the latter of these two distributions is ‘masked out’, by the zero value in assumption (6), from the distributions (3,5,13), so that parameters a', b' disappear with ξ from the ‘ideal observations’ model in that case. See columns 5 and 6 (of 8) in Table 1 on p43. In other cases, in which $\xi > 0$, note that our assumption (10) translates into a messy but computable constraint on $a, b, a', b', p_{0i}, p_{0c}$. (One may simply use analytic differentiation w.r.t. s , and then numerical zero-finding of monotonic functions, to determine the local minima of RHS–LHS in inequality (10), which, with the limiting values at the two end-points, $s \rightarrow 0, 1$ can be used to formulate the constraint that the inequality shall hold over the whole interval $0 \leq s < 1$.)

4.3 Effect of Node Probability Table Assumptions on Equation (5)

The above assumptions about the node probability tables can now be substituted in the *numerator of (5)*. This numerator is in fact simply the prior probability density of “*S* and *ideal observation on both argument legs*”. We will denote it $P(S\&ideal\ obs.)$, meaning, more precisely

$$P(s\&ideal\ obs.) = P((C, S, V, T)=(accepted, s, verified, no\ failures)). \quad (13)$$

Table 1 on p43 shows how the value of (13) is a sum of four terms corresponding to the different configurations of ZO — dealing separately with the case $S=0$. The four terms summed are each a product of three entries in the top part of a vertical column of the table (in each single column, the three entries in the three rows immediately under the double line that separates the headers) weighted also by the prior probability of the value of the pair ZO which selects the column. Without this $P(ZO)$ weighting factor, the column-product

$$P(VTS | ZO) = P(T | OS)P(S | Z)P(V | SZ)$$

is the probability (density, for $S>0$) of seeing the ideal observations VT , *and* of the pfd having a true (unknown) value S , conditioned (as if these were known) on specified ZO values. Note that the case $s>0$ becomes much simplified under the verification infallibility assumption $(\alpha, \xi) = (0, 0)$, with the two zero ξ values in the third row of the body of the table causing some terms from subsequent rows to disappear: if we assume it impossible to verify an imperfect system against a correct specification, then a part of the total probability (13) disappears. The second to last row of the table gives the value of (13), for $S=0$ on the left, and for $0<S\leq 1$ on the right. In the latter case, this probability is actually a density in its S -argument. We can think of the value “*ideal obs.*” as the vector of observed values of CVT , or effectively of just VT , since we have assumed C to be determined by VT in this ideal case (4).

For these *ideal observations* $VT = (verified, no\ failures)$, the $S=0$ -cases of the three equations (7-9) produce the six¹¹ 1’s that occur in the left half of Table 1. These assumptions therefore mean that $P(ideal\ obs. | S=0, Z=incorrect, O) = 1$, and $P(ideal\ obs. | S=0, Z=correct, O) = 1-\alpha$. I.e., under our conservative assumptions, and irrespective of assumed correctness or otherwise of the oracle, the conditional probability of seeing the ideal observations from a system *known to be perfect* becomes certainty, if the specification is assumed *incorrect*, and $1-\alpha$ if the specification is assumed correct.

¹¹ Note how just three equations produce six 1’s here essentially because the topology says that $T \perp Z | OS$ and $V \perp O | ZS$. So, reading across the first row, the conditional probabilities of T alternate, in each half of the table; and reading across the third row, the conditional probabilities of V occur in adjacent pairs.

The last row of Table 1 gives the *denominator of (5)*. This normaliser is just $P(\textit{ideal obs.}) = P((V,T)=(\textit{verified,no failures}))$. Substituting the entries of Table 1 into (5), gives the conditional probability $P(S=0 \mid \textit{ideal obs.})$ and also, for $S>0$, the conditional probability density pdf($S \mid \textit{ideal obs.}$), in each case as the ratio of expressions in the last two rows of the table. Care is necessary in use of notation for discontinuities and points of concentrated mass: Note the comment under the table.

5 Expressions for Confidence and Doubt

Substituting these further assumptions and abbreviated notations into the penultimate row of Table 1 on p43 (which originated from the numerator of (5)) gives the concentrated mass

$$P(S=0 \ \& \ \textit{ideal obs.}) = (1-\alpha)p_{0c}\pi_{c*} + p_{0i}\pi_{i*} \quad (14)$$

and, for strictly positive values of S , the pdf

$$\begin{aligned} \text{pdf}(s \ \& \ \textit{ideal obs.}) = \\ \xi \frac{(1-p_{0c})}{\beta(a',b')} s^{a'-1} (1-s)^{b'-1} [\pi_{cc}(1-s)^n + \pi_{ci}] + \frac{(1-p_{0i})}{\beta(a,b)} s^{a-1} (1-s)^{b-1} [\pi_{ic}(1-s)^n + \pi_{ii}], \\ 0 < s \leq 1. \end{aligned} \quad (15)$$

To obtain the conditional distribution of S given the ideal observations, we require a normaliser from the above joint probability density (and point mass) corresponding to the last row of Table 1

$$P(\textit{ideal obs.}) = P(S=0 \ \& \ \textit{ideal obs.}) + \int_{0 < s \leq 1} \text{pdf}(s \ \& \ \textit{ideal obs.}) ds \quad (16)$$

where the left-hand term is the point mass contribution, which is understood to be excluded from the domain of the right-hand, integral term. Substituting from (14) and (15) yields

$$P(\textit{ideal obs.}) = (1-\alpha)p_{0c}\pi_{c*} + p_{0i}\pi_{i*} + \xi(1-p_{0c}) [\pi_{cc}\mu' + \pi_{ci}] + (1-p_{0i}) [\pi_{ic}\mu + \pi_{ii}] \quad (17)$$

using notation μ for the n^{th} non-central moment of the beta distribution¹²,

$$\mu = \frac{\beta(a, b+n)}{\beta(a, b)}, \quad \mu' = \frac{\beta(a', b'+n)}{\beta(a', b')}$$

¹²—to be precise, of the beta distribution with its usual parameters a and b interchanged, because $\{1-X \text{ is distributed Beta}(b, a)\} \Leftrightarrow \{X \text{ is distributed Beta}(a, b)\}$.

in order to shorten some expressions in what follows. Note the dependence of μ and μ' on n , as well as on the beta distribution parameters. For any fixed $a, b, a', b' > 0$, we always have μ, μ' strictly decreasing in n , both being 1 at $n=0$, and having $\mu, \mu' \rightarrow 0$ as $n \rightarrow \infty$ (though the convergence can be slow for small a, a').

From (15) and (17), we can express the *doubt*, or probability of ‘unsafe failure’ of the entire two-legged assessment procedure represented by these modelling assumptions, with the formula

$$P(S > s \mid \text{ideal obs.}) = \frac{\xi(1-p_{0c}) [\pi_{cc}\mu' I_{1-s}(b'+n, a') + \pi_{ci} I_{1-s}(b', a')] + (1-p_{0i}) [\pi_{ic}\mu I_{1-s}(b+n, a) + \pi_{ii} I_{1-s}(b, a)]}{(1-\alpha)p_{0c}\pi_{c*} + p_{0i}\pi_{i*} + \xi(1-p_{0c}) [\pi_{cc}\mu' + \pi_{ci}] + (1-p_{0i}) [\pi_{ic}\mu + \pi_{ii}]} \quad (18)$$

where I_{1-s} denotes the (regularised) *incomplete beta function* using the notation

$$I_x(a, b) = \frac{1}{\beta(a, b)} \int_0^x u^{a-1} (1-u)^{b-1} du, \quad a, b > 0, 0 \leq x \leq 1, \quad (19)$$

which is a strictly increasing function of $x \in [0, 1]$ for all $a, b > 0$. [See [20, p944] for its other properties, which include $I_0(a, b) = 0$, $I_1(a, b) = 1$, and $I_{1-x}(a, b) + I_x(b, a) = 1$.] Equation (18), with the strict inequality in the probability on the left hand side, actually holds for all (non-negative) s , including $s=0$.

We shall refer to the conditional probability (18) in what follows as the ‘doubt function’. This function of 13 independent arguments (the threshold s value; and the 12 independent parameters of our node conditional probabilities) is an important consequence of our model as it stands, capturing the probability of the most important kind of ‘argument failure’ we first identified on p5. There are several related measures that could be substituted here. To be exact, this one is the *conditional* probability that such an unsafe argument failure has occurred, given that a system is deemed *accepted* by the two-legged argument. Of course this is distinct from the unconditional probability that a randomly selected system will be truly unsafe ($S > s$) and will be (incorrectly) accepted by the two-legged argument as sufficiently safe (the *numerator* of (18)). It is also different from the probability that a randomly selected *unsafe* system will be deemed sufficiently safe by the two-legged argument. The conversions between these are straightforward, depending on quantities such as the ‘base rate’ of truly unsafe systems among systems which are submitted for evaluation by this procedure, and the rates at which rejections of randomly submitted systems will occur. Note that in our model as presently formulated, these marginal background rates are not outside the scope of the model. They *are* implied by our chosen values for model parameters: in the first case (the marginal probability $P(S > 10^{-3})$) by π 's, and $p_{0i}, a, b, p_{0c}, a', b'$;

and in the latter case (the marginal probability $P(C=rejected)$) by the entire set of 12 independent model parameters.

We note again the significant degree of simplification occurring in the case $\xi=0$. In that case, not only do parameters a', b' become irrelevant to the posterior probability distribution (18) of S given the ideal observations, but also (18) depends on only 2 of the 3 independent degrees of freedom of the prior ZO distribution π . The conditional probability (18), in this special $\xi=0$ -case of the *ideal observations* scenario, depends on the marginal probability $P(Z)$ and on the conditional probability $P(O|Z=incorrect)$. Its lack of dependence on $P(O|Z=correct)$ is explained by the fact that the infallibility assumption $\xi=0$ for the verification process in the case ($S>0, Z=correct$), given by the top line of (6) creates two zeros in the 5th and 6th columns of Table 1 which, by multiplication, effectively ‘mask out’ from the final rows of Table 1 (i.e., from (2) and the numerator and denominator of equations (3) & (5)) the *only* case of dependence of any term in these equations on the state of variable O when $Z = correct$ (the pair of unequal entries in Table 1 located two rows above this pair of zeros). The fact that this masked-out ($S>0, Z=correct$) scenario *is* the only means, under ideal observations, by which our posterior distribution of S could otherwise (without this masking effect of assumption (6)) depend on the state of O given $Z=correct$ relies on the conservative assumption (9) which removes any dependence on O in the LHS of Table 1 where $S=0$. As soon as we relax either the verification infallibility, $\xi=0$, or the conservative assumption (9) which interact in this simplifying way here, we obtain the greater complexity of a conditional probability $P(S>s | ideal obs.)$ which involves all 12 independent parameters of our parametric node probabilities.

If s is small, it may be numerically more accurate (depending e.g. on the precision properties of the incomplete beta algorithm at its extreme arguments) to compute instead the *confidence* using

$$\begin{aligned}
P(S \leq s | ideal obs.) &= \\
P(S=0 | ideal obs.) + P(0 < S \leq s | ideal obs.) &= \\
&\frac{(1-\alpha)p_{0c}\pi_{c*} + p_{0i}\pi_{i*}}{(1-\alpha)p_{0c}\pi_{c*} + p_{0i}\pi_{i*} + \xi(1-p_{0c})[\pi_{cc}\mu' + \pi_{ci}] + (1-p_{0i})[\pi_{ic}\mu + \pi_{ii}]} + \\
&\frac{\xi(1-p_{0c})[\pi_{cc}\mu' I_s(a', b'+n) + \pi_{ci} I_s(a', b')] + (1-p_{0i})[\pi_{ic}\mu I_s(a, b+n) + \pi_{ii} I_s(a, b)]}{(1-\alpha)p_{0c}\pi_{c*} + p_{0i}\pi_{i*} + \xi(1-p_{0c})[\pi_{cc}\mu' + \pi_{ci}] + (1-p_{0i})[\pi_{ic}\mu + \pi_{ii}]}
\end{aligned} \tag{20}$$

To look briefly at some actual numbers, we will consider first an infallible verification assumption of the form $(\alpha, \xi) = (0, 0)$, which reduces the number of active model parameters significantly, from 12 to 7 for the reasons just explained. Under this simplifying assumption, all entries in the third row of Table 1 on p43 are zeros and ones. These eight entries in fact – given the CI as-

sumptions of the model topology – may be taken in pairs, reading across, and actually represent only four CI tables entries (because of the lack of dependence on variable O , the oracle correctness). The 3rd, 4th, 7th, and 8th, entries in this row of the table are justified as a single conservative assumption to cover the case of an incorrect system specification. Here we have assumed pessimistically, with equation (7), that the verification against such an incorrect yardstick will always produce a positive conclusion about any built system, irrespective of its actual failure probability S . We will retain this conservative assumption below. In contrast, for the other case—that the specification Z is correct—the remaining four entries of the same row are affected by two kinds of ‘infallibility’ assumption, $\alpha=0$ and $\xi=0$, for the verification activity. We cannot justify these from an argument for conservatism. The assumption $\xi=0$ affecting the 5th and 6th columns can even be viewed as over optimistic, depending on how we define correctness of a specification, and how the verification is carried out in practice.

If, under this $(\alpha, \xi) = (0, 0)$ assumption, we set¹³

$$(p_{0i}, a, b, p_{0c}, n, \pi_{c*}) = (0.2, 1, 999, 0.5, 4602, 0.8). \quad (21)$$

in (18), we obtain doubt

$$P(S > s | \text{ideal obs.}) = \frac{\pi_{ii} I_{1-s}(999, 1) + 0.178 \pi_{ic} I_{1-s}(999+4602, 1)}{0.55 + \pi_{ii} + 0.178 \pi_{ic}}$$

where the incomplete beta term $I_{1-s}(999, 1)$ may be thought of as the corresponding probability $P(X > s)$ for a random variable X which is beta distributed with parameters $(a, b) = (1, 999)$,

$$P(X > s) = \frac{\int_s^1 u^{a-1} (1-u)^{b-1} du}{\beta(a, b)} = 1 - I_s(a, b) = I_{1-s}(b, a).$$

The same applies to the other incomplete beta term, but instead with parameters $(a, b) = (1, 5601)$. The latter is a smaller beta probability – considerably smaller for many s -values: The respective means of these two beta distributions being 0.001 and 0.00018. Under our assumption $\xi=0$, it makes no difference to (18) how the probability of specification correctness, $\pi_{c*}=0.8$ is divided between the probabilities of $(Z, O) = (\text{correct}, \text{correct})$ and $(Z, O) = (\text{correct}, \text{incorrect})$. However, the distribution of the other 20% of prior belief,

¹³ An interesting figure to use for the number of test-cases in (8) is $n = 4,602$. This is the number of tests required to give a Bayesian 99% upper confidence bound, on the pfd S , that is less than 10^{-3} when no failures are observed, *under the simpler model assumptions used in [21]*. In terms of our model here, those assumptions say essentially that there is no verification leg, that the oracle is known to behave perfectly, and that the prior distribution of Z and the conditional distribution $P(S|Z)$ are such that S is initially uniformly distributed on the unit interval.

$\pi_{i*} = 0.2$, does make a difference, and in fact can be used to change the doubt considerably as it is differently allocated between $(Z, O) = (\textit{incorrect}, \textit{correct})$ and $(Z, O) = (\textit{incorrect}, \textit{incorrect})$. It is easy to see that the two extreme cases $(\pi_{ic}, \pi_{ii}) = (0.2, 0)$ and $(\pi_{ic}, \pi_{ii}) = (0, 0.2)$ result in values for doubt

$$P(S > s | \textit{ideal obs.}) = \frac{0.178 \times 0.2 I_{1-s}(5601, 1)}{0.55 + 0.178 \times 0.2}, \quad \text{and} \quad \frac{0.2 I_{1-s}(999, 1)}{0.55 + 0.2},$$

respectively, while values of (π_{ic}, π_{ii}) between these two extremes lead to values intermediate between these two, producing a curve which is hyperbolic in form and monotonic increasing¹⁴ as the 0.2 probability mass is steadily shifted from π_{ic} to π_{ii} . For instance, if we put $s=10^{-3}$, these two end points of this increasing hyperbolic segment are $P(S > s | \textit{ideal obs.}) = 0.00022$ at $\pi_{ic}=0.2$, and $P(S > s | \textit{ideal obs.}) = 0.098$ at $\pi_{ii}=0.2$

These numbers illustrate how the joint prior beliefs concerning the two unobservable variables ZO – incorporating both the prior doubt about Z and O individually, as well as beliefs about the *association* between their likely values – can influence the doubt about the claim produced by the two-legged argument. Although the model we are using is very simplified at this stage, we expect that such prior beliefs represented at ‘the top part’ of our BBN topology may well continue, under more sophisticated and realistic models, to be a driver for the amount of extra confidence gained when safety arguments are combined in this kind of formal way.

6 How Effective is this Multi-Legged Argument Approach in Gaining Confidence in Dependability Claims?

We have mentioned earlier that the use of multiple argument legs arises informally from reasoning similar to that used to justify the use of diverse channel redundancy to obtain system reliability. Questions that are of interest for systems have similar counterparts here. How much of a confidence gain do we obtain, via the above two-legged argument model, over the verification-only argument model, or the testing-only argument model? Equation (18) expresses the monotonic ‘doubt function’ $P(S > s | \textit{ideal obs.})$ of s as a function of the twelve independent model variables: $\pi_{ic}, \pi_{ii}, \pi_{cc}, p_{0c}, p_{0i}, a, b, a', b', \alpha, \xi, n$: how can we best gain an understanding of the ‘shape’ of this functional dependence? What are the important drivers of the benefit coming from the use of multiple legs? E.g. does correlation between doubts in the assumptions play an important role, as intuition would suggest? How does the two-legged argument doubt (18) compare with its ‘naive independence’ version in which,

¹⁴ assuming $s \neq 0, 1$

for a fixed claim, the doubts emanating from the two single argument legs are simply multiplied to produce the supposed two-leg-argument doubt?

6.1 *Benefit of the Two-legged Argument and its Dependence on Stochastic Association Between Doubts in Assumptions for each Leg*

We show next that it is possible to assign the parameters of our two-legged model such that we leave only one value possible, either of V , or of T , respectively, or of each of V and T . This single value is, in each case, the “ideal observation” value $V=verified$ or $T=no failures$. Thus, we can choose parameters which make, in (2) and the equations following, $P(V=verified|SZ)$, or $P(T=no failures|OS)$, respectively, become a constant, 1, (so no longer depending on the values of SZO). Note that this procedure of altering the model’s local conditional probability tables until only one value of a variable (of V , or of T , here) has positive probability is mathematically distinct from summation over that variable (with tables such that two or more of its values have positive probability). It is easy to verify in the case of our model that the first procedure does in fact produce a factorization of the joint distribution which – at least in the cases of “ideal observations” – is identical to that which would arise from one or other of our proposed single argument topologies Figures 2 and 3 on p10. More precisely stated: although it is possible to derive, for each of the two single-legged arguments of Figs. 2 & 3, a parametric representation of the ‘doubt’ (18), just as we have done above for the two-legged argument, it is actually simpler, and in fact equivalent (in the case where we condition on ideal observations), to deduce directly the analogous consequences of the single-leg arguments of Figures 2 & 3 as *special cases of the results for the combined argument* obtained by the following special parametric assignments.

Our removal of testing evidence from the argument is equivalent to substituting $n=0$ in (18) to produce a doubt

$$P(S>s | V=verified) = \frac{\xi(1-p_{0c})\pi_{c*}I_{1-s}(b', a') + (1-p_{0i})\pi_{i*}I_{1-s}(b, a)}{[(1-\alpha)p_{0c} + \xi(1-p_{0c})]\pi_{c*} + \pi_{i*}} \quad (22)$$

for the verification-only argument. (Unsurprisingly, the initial beliefs about the likely oracle-correctness are not present in this expression.)

Similarly, for the testing-only argument, we can substitute $(\alpha, \xi)=(0, 1)$ in (18)

to produce

$$P(S > s | T = \text{no failures}) = \frac{(1-p_{0c})[\pi_{cc}\mu' I_{1-s}(b'+n, a') + \pi_{ci} I_{1-s}(b', a')] + (1-p_{0i})[\pi_{ic}\mu I_{1-s}(b+n, a) + \pi_{ii} I_{1-s}(b, a)]}{p_{0c}\pi_{c*} + p_{0i}\pi_{i*} + (1-p_{0c})[\pi_{cc}\mu' + \pi_{ci}] + (1-p_{0i})[\pi_{ic}\mu + \pi_{ii}]} \quad (23)$$

for our *ideal observations* case of the single, testing-argument leg. To see this¹⁵, consider the third row (of 5 rows) in the body of Table 1 on p43. The above substitutions make all entries in this row equal. That is to say, the substitutions make $P(V = \text{verified} | ZS)$ independent of the values of both Z and S , in just the manner we described at the beginning of this subsection. (Consider the effect on equations (3,5).) It is not difficult to confirm from our algebraic conclusions that the effect of this is equivalent to the removal of node V from the model, i.e. the conversion of Figure 1 to Figure 2, as required.

We now compare some plots of the three doubt function (18, 22, & 23), for fixed $s=10^{-3}$, as we vary certain other model parameters. The plots of Figure 4

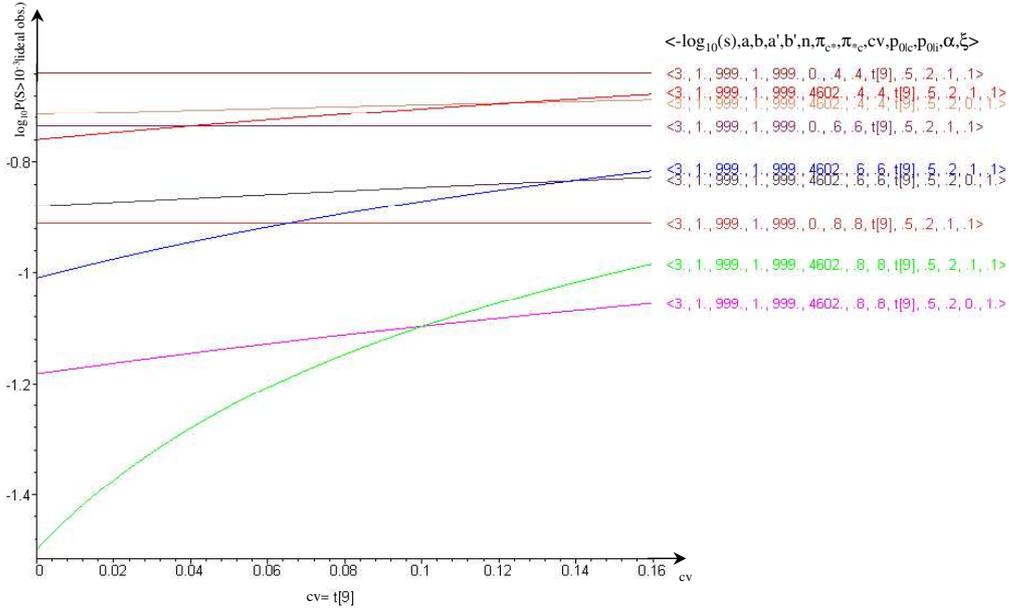


Fig. 4. Doubt functions (18) vs. cv – with params. as in (24) and following text.

show the value of \log_{10} of the doubt function (18) and were produced by setting the threshold s value to 10^{-3} and the model parameters

$$(p_{0i}, a, b, p_{0c}, a', b') = (0.2, 1, 999, 0.5, 1, 999). \quad (24)$$

The nine plots shown may be thought of as three groups of three. These groups

¹⁵ This $(\alpha, \xi) = (0, 1)$ substitution procedure would not produce the single testing leg model if we relaxed the conservative assumption of equation (7) .

were produced by setting (n, α, ξ) in turn to the values $(4602, 0.1, 0.1)$, then $(4602, 0, 1)$, then $(0, 0.1, 0.1)$. The *first* of these three vectors represents a *two-legged argument*, which may be compared against *the second two* which are the special cases (23) and (22) of (18), corresponding respectively to *testing-only*, and *verification-only, single-legged arguments*, with the other parameters (24) common to all three groups.

Within each group of three, the individual plots are distinguished, and the variation of each along the horizontal axis is determined, by manipulating the π -matrix as follows. π was reparameterised in terms of the triple (π_{c*}, π_{*c}, cv) . The first two of these three parameters are the prior marginal probabilities of correctness for the specification Z and the oracle O . Variation of these is what separates the three different plots within each group. The third parameter cv , used as the horizontal axis, is an analog of the *covariance* between Z and O (not strictly a covariance in the usual sense since the state-spaces of these two variables are not numeric, but equal to the covariance if they were converted to a pair of Bernoulli random variables).

$$\pi = \begin{bmatrix} \pi_{cc} & \pi_{ci} \\ \pi_{ic} & \pi_{ii} \end{bmatrix} = \begin{bmatrix} \pi_{c*}\pi_{*c} + cv & \pi_{c*}\pi_{*i} - cv \\ \pi_{i*}\pi_{*c} - cv & \pi_{i*}\pi_{*i} + cv \end{bmatrix}.$$

Thus *each individual plot* considered alone represents the effect on the value of the doubt function of an increasing positive, prior *covariance* cv between *the correctness of specification and of oracle*, while keeping the prior, *marginal* correctness probabilities of both Z and O fixed. *Comparison between* the 9 plots illustrates both the effect of changing marginal correctness probabilities π_{c*} and π_{*c} , as well as the comparison of the two single-legged arguments, against each other, and against the two-legged argument. It perhaps helps to disentangle the plots in Figure 4 to state the following descriptive observations about the shapes of the 9 curves. We will refer to individual curves here by numbering them **1. . . 9**, counting vertically upwards at the right-hand end of the graph, with the lowest plot numbered **1**. (Because of some curves crossing, this means that the plots at the left-hand end of the graph, are numbered in the order **2, 1, 5, 3, 4, 8, 6, 7, 9**, moving upwards):

- The three flat, horizontal plots (**3, 6, 9**) are the doubt functions for the verification-only argument. They are flat because the verification-only argument does not depend on the correlation cv of prior correctness of specification Z and oracle O . It depends only on the prior marginal distribution π_{c*} of Z (22).
- The prior marginals (π_{c*}, π_{*c}) have the values $(0.8, 0.8)$ in plots (**1, 2, 3**), $(0.6, 0.6)$ in plots (**4, 5, 6**), and $(0.4, 0.4)$ in plots (**7, 8, 9**). For each fixed value of the prior marginals, towards the left-hand end (small cv) of the

graph¹⁶, there is a clear ordering in ‘performance’ (for these particular chosen model parameter values) between the three arguments: two-legged argument (**2, 5, 8**) is better than testing-only argument (**1, 4, 7**) which is better than verification-only argument (**3, 6, 9**). However the ordering of the two-legged, as compared to the testing-only is reversed towards the right-hand end of the graph. (See further discussion below.)

- There are six plotted doubt functions which are not flat. Of these, three (**2, 5, 8**) are from the two-legged argument, and three (**1, 4, 7**) are from the testing-only argument. For each π_{c^*} , and π_{*c} , the two-legged argument doubt function has a steeper gradient – as a function of the correlation cv – than the corresponding testing-only argument. (**2** is steeper than **1**, etc.) The latter does increase as a function of cv , but, in each case, rather more gently than it would with the addition of verification evidence.
- For arguments of each of the three kinds, doubt always increases (confidence diminishes) as the prior marginal probabilities of correctness of Z and O decrease (from $(\pi_{c^*}, \pi_{*c}) = (0.8, 0.8)$ in curves (**1, 2, 3**), down to $(0.6, 0.6)$ in (**4, 5, 6**), down to $(0.4, 0.4)$ in (**7, 8, 9**)).

For a numerical example of the ordering observed in the second bullet point above, if we fix $cv = 0.06$, we obtain prior joint ZO distributions

$$\pi = \begin{bmatrix} 0.7 & 0.1 \\ 0.1 & 0.1 \end{bmatrix}, \quad \text{or} \quad \begin{bmatrix} 0.42 & 0.18 \\ 0.18 & 0.22 \end{bmatrix}, \quad \text{or} \quad \begin{bmatrix} 0.22 & 0.18 \\ 0.18 & 0.42 \end{bmatrix}$$

for the correctness of the oracle and specification, accordingly as the marginal correctness probabilities π_{c^*}, π_{*c} are assigned values 0.8, or 0.6, or 0.4. These three π -matrices result, respectively, in values of our doubt function, ordered as (*verification-leg, testing-leg, two-legged*), of $(0.12, 0.074, 0.062)$, or $(0.18, 0.14, 0.12)$, or $(0.23, 0.20, 0.19)$. Notice that the 2-legged argument gives the greatest confidence in each case.

In comparing the two-legged argument with the testing-only, single-legged argument, one notable observation is that, as the correlation cv between specification and oracle correctness becomes very (perhaps implausibly?) high towards the right-hand sides of the plots, we seem to arrive at a situation in which the fact of being informed that the system has been successfully verified against its specification slightly *undermines* the high confidence that had been obtained from the failure-free testing alone. This is the first of several, at first sight, counter-intuitive model behaviours that we shall meet. We see below that our current model topology, with its uncertainties concerning a potentially defective oracle or specification and use of conservative assumptions,

¹⁶ but continuing, as before, to refer to the plots by number, according to their order at the *right-hand* end

allows some complex kinds of model behaviour which may either be realistic features of rational uncertainty, or only spurious model artifacts. In the latter case their exclusion may require parameter constraints on the node probability distributions with which, it will perhaps eventually be possible to argue, all competent expert beliefs will necessarily conform.

The examples above all involve a symmetric π -matrix, so that Z and O have equal marginal probabilities of being correct. Of course we have no reason to suppose this is representative of real beliefs for this kind of system. We experimented with various other parameter values and obtained a varied set of conclusions as to the comparative efficacy of the two-legged and single-legged arguments, based on this model. To examine one more numerical example, which has a slightly higher prior ‘‘covariance’’ $cv = 0.075$ between Z and O correctness, but coupled with some rather more optimistic values of other parameters, and a greater prior confidence in specification correctness than in oracle correctness, put

$$\pi = \begin{bmatrix} 0.25 & 0.40 \\ 0.25 & 0.10 \end{bmatrix}.$$

We shall express a rather high prior confidence in the reliability of the verification process against a correct specification $\alpha=0.01, \xi=0.04$, and slightly more optimistic prior beliefs than above about the likely values of S when Z is incorrect,

$$(p_{0i}, a, b, p_{0c}, a', b') = (0.4, 1, 999, 0.5, 1, 999), \quad (25)$$

and retain the same values as above for the amount of testing $n=4, 602$ and the claim $S \leq s=10^{-3}$. Using these values produces approximately equal confidence in the claim from each single leg, and almost a two thirds reduction in doubt when we use our model to combine the evidence from these two separate legs. The doubt function values $P(S > 10^{-3} | \text{ideal obs.})$, in the order (*verification-leg, testing-leg, two-legged*), are (0.12, 0.12, 0.045). So, as might be expected, a two-legged argument *can* bring considerable increase in confidence about a claim, in comparison with each of its constituent legs. Whether this occurs in practice will, of course, depend upon the details of the expert beliefs as represented by the model parameters.

6.2 Doubt as a function of the number n of test cases

Under ‘*ideal observations*’ (no detected failure), one might expect, as with earlier models [21], that S should stochastically decrease – in terms of its posterior distribution (18), or (23) in the testing-only case – monotonically as the quantity n of positive testing evidence accumulates. Increasing confidence from continued failure-free testing ought surely to make this posterior

probability decrease monotonically in n , for every fixed s .

We will not attempt to solve for the general parametric assumptions required such that our two-legged argument, with testing evidence incorporated, should provide higher confidence than the single, verification-only leg (the $n=0$ case (22) of (18)). We only note that for ‘very large’ n this is so whenever (22) exceeds (18) with 0 substituted for μ and μ' . This is because of the limiting property of the beta moments noted on p18 (recalling that the incomplete beta terms in (18) are bounded by 1 as $n \rightarrow \infty$). This is the case in all of the

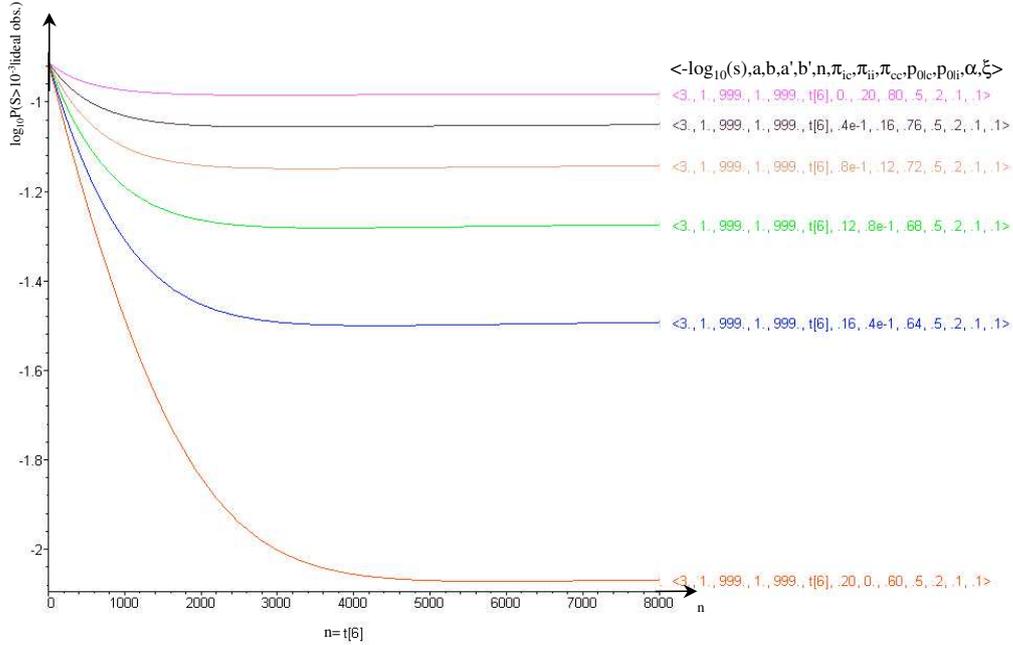


Fig. 5. Doubt functions (18) vs. n – with params. as in (26)

example plots of Figure 5. These show the value of \log_{10} of the doubt function (18) plotted against the number n of test cases, using a claim $s=10^{-3}$, and model parameters

$$(p_{0i}, a, b, p_{0c}, a', b', \alpha, \xi) = (0.2, 1, 999, 0.5, 1, 999, 0.1, 0.1). \quad (26)$$

The plots differ from each other only in the values of the π matrix, which were produced by fixing the marginal probabilities of specification/oracle-correctness at $\pi_{c*}=\pi_{*c}=0.8$. The correlation coefficient between Z - and O -correctness varies as we read down the key on the RHS, with an unbelievable exact equality in the top plot (specification is correct precisely and only when oracle is correct; otherwise, both are incorrect), working down to an almost equally unbelievable negative correlation at the bottom. The second to bottom plot represents independence ($cv=0$) between the correctness of the oracle and of the specification. Clearly the general pattern here is again that low (or even negative) correlation of the prior beliefs in specification and oracle correctness yields an advantage in terms of confidence levels derivable from the two-legged

argument, confirming our observation of the same tendency in Figure 4 above. Notice that there appears to be a very slight deterioration in confidence at large n values, i.e. there is a turning point at some n -value (which varies from one plot to another) occurring before the limiting $n \rightarrow \infty$ value is approached. We give some explanation of this quirk in the next section, and of other model characteristics which might appear counter-intuitive on first inspection.

7 Some Counterintuitive Results

7.1 Supportive and Non-Supportive Single Argument Legs

As we have seen in the previous section, the acquisition of evidence that is ‘obviously good news’ – what we have called ideal evidence – can sometimes result in a *reduction* in confidence. In this section we shall examine in some detail examples of such apparently non-intuitive results. Our approach is to use some numerical search and optimization tools to investigate the question of whether or not, under the present model, the arrival of ideal ‘supportive’ evidence results in increased confidence in the pfd.

We begin with a *single* argument. At first sight it might appear self-evident what ‘supportive evidence’ should look like: For a verification leg, the evidence is supportive precisely if the system is *verified correct*. For a testing leg (though there may also be a grey area), it seems at first sight clear that *no failures*, or very few failures, amongst a large number of test cases is supportive evidence. In contrast, a system’s failure to be verified correct, or to succeed sufficiently often during test, should reduce our confidence.

In fact we can show that this is not always the case. Consider, for example, a testing argument leg in which no failures have been observed among the test cases, and the parameter values of the model are: $s=0.001$, $n=17,921$, $a=2.58276$, $b=4.77020$, $a'=16.68483$, $b'=41,133.7$, $p_{0i}=2.00200 \times 10^{-3}$, $p_{0c}=4.21724 \times 10^{-3}$, and

$$\pi = \begin{bmatrix} 0.994192 & 1.63910 \times 10^{-3} \\ 7.81537 \times 10^{-5} & 4.09042 \times 10^{-3} \end{bmatrix}.$$

The beliefs about the unobservable variables ZO are changed by this evidence from the above prior π to

$$P(ZO|T) = \begin{bmatrix} 0.53406 & 0.13329 \\ 1.2724 \times 10^{-5} & 0.33263 \end{bmatrix}.$$

The prior confidence in the claim here is 0.99583, but the posterior confidence, in spite of the extensive ‘good news’ from testing, is *decreased* to 0.66803. At

first glance such a result is quite surprising and counter-intuitive. On closer inspection, however, we believe there is an intuitive explanation in terms of two ‘rival explanations’ for a long period of failure-free testing. Reasoning informally, guided by the property of adjacency in the topology of Figure 2 on p10, we see that on the one hand extensive failure-free testing, applied as evidence to node T , may be taken to indicate a low system pfd S . Realistic node conditional probabilities for T will surely capture this effect. But, symmetrically on the left-hand side of Figure 2, a parallel *negative* inference of a *potentially low oracle quality* O also suggests itself, as a rival explanation of the apparently positive testing evidence: perhaps a defective oracle is missing failures. These two inferences from T may *both* follow when n becomes very large without failure. One can imagine a tension arising at node S between these two competing tendencies of the testing evidence. This raises the question of how the two inferences interact, and whether one ‘dominates’ in its effect upon posterior beliefs about S . The answer will depend partly on the conditional probability distribution $P(T|OS)$. (Note that – while in no sense *essential* to this explanation – the ‘conservative’ assumption (9) would appear to increase the viability of the second potential explanation for apparently successful testing.) But our model in Figure 2 also contains a prior belief structure ‘higher up’, usually with stochastic association of some kind between the variables ZOS . The combined effect of the two available inferences from observing large n without failure will depend also on this ‘upper part’ of the topology. In particular, does this structure of prior belief contain a positive prior probabilistic association between correctness of the oracle O and quality of the system (small value for S) – perhaps via a shared association with the correctness of Z ? If the answer to this question is “yes” then it seems plausible that, for certain model parameter values, *evidence of a defective oracle could have the effect of reducing confidence* in a low system pfd. E.g., in the last numerical example the increased doubt in the correctness of the oracle (sum of terms in last column of matrices above) is associated with an increased doubt about the correctness of the specification (sum of terms of last row of the matrices). Thus when we see very many failure-free test cases we may increase our mistrust in the oracle, increase our mistrust in the specification *and thus increase our mistrust in the pfd* – see constraint (10). We stress that this informal conception of competing effects at S of successful testing is not intrinsic to the topology of Figure 2, but relies heavily on node conditional probability assumptions. The model topology is sufficiently flexible to allow such possibilities, but whether or not they are found will depend also on the 2×2 prior matrix π for ZO , the parameter values used for our two point-mass-augmented beta distributions (12) for $P(S|Z)$, and on the conditional probabilities (8) & (9) comprising $P(T|OS)$.

Examples of non-supportive verification arguments may also be obtained. These might be explained by a similar inference concerning the likely correctness of Z , whenever the model parameter values are such as to create

the needed associations. For example, a successful system verification with $s=0.001$, $a=1.2742$, $b=0.2106$, $a'=3.2095$, $b'=27,095$, $\alpha=0.3950$, $\xi=1.2006 \times 10^{-4}$, $p_{oi}=1.5547 \times 10^{-4}$, $p_{oc}=1.3812 \times 10^{-3}$, $\pi_{c*}=0.9997156$, $\pi_{i*}=2.844 \times 10^{-4}$ gives a decrease from a high prior confidence 0.99972 to the much lower 0.77064 after the positive verification. We suggest that while this, perhaps counter-intuitive, behaviour may well have been accentuated by our conservative assumption (7) (for the verification outcome in the case that the specification $Z=incorrect$), it would still be possible after a degree of relaxation of this conservative assumption, given the ‘right’ conditional probabilities applied to our topology. In more detail, our tentative explanation for this effect is as follows: While ignorant of the true value of S , the observation of a successful verification at V provides, under our conservative assumption, *stronger support for the incorrectness, than for the correctness, of the specification Z* . Specifically, beliefs about the unobservable variable Z become changed (by the news of the successful system verification) from the prior probability 0.9997156 of Z ’s correctness to $P(Z=correct | V) = 0.77060$. This opens a chain of subsequent inference of the following kind. The prior *stochastic association between correct Z and small S* required by (10) gives this *increased Z -doubt* a tendency to increase the probability of large S .

In the case of the numbers used above, we can perhaps best illustrate this by dividing the range of S up into two bins $[0, 10^{-3}]$ and $(10^{-3}, \infty)$. Using the same 2×2 matrix layout that we have used above for joint beliefs about ZO , we can compute the prior beliefs

$$P(ZS) = \begin{bmatrix} 0.99971563 & 1.09403 \times 10^{-9} \\ 5.18016 \times 10^{-8} & 2.84319 \times 10^{-4} \end{bmatrix}.$$

which quantify (an aspect of) the association between Z and S mentioned above. Compare the posterior counterpart of these beliefs, having observed the ideal verification evidence

$$P(ZS | V) = \begin{bmatrix} 0.77060 & 1.05960 \times 10^{-10} \\ 4.17888 \times 10^{-5} & 0.22936 \end{bmatrix}.$$

which illustrates how the doubt cast on Z by the successful verification has caused belief to ‘shift along the main diagonal’ from $(Z=correct, S \leq 10^{-3})$ to $(Z=incorrect, S > 10^{-3})$.

We do not assert that the chains of inference shown in these examples will be realistic in every model application. Much will depend on the parameter values assigned to the node conditional probabilities, and how, for example, these stochastically associate variables Z , O , and S . We might well later choose to replace the assumptions we have called ‘conservative’ by alternatives that we

might elicit as real experts' beliefs, about the likely effects of defective oracles and specifications. However, our examples of quite subtle interactions between evidence, assumptions, and assumption doubt do illustrate the richness of the kinds of competing inferences modelled within even a highly simplified BBN structure, and alert us to the fact that symbolic analysis may identify model consequences that initially will seem surprising and counter-intuitive.

7.2 Adding a Supportive Leg May Not Improve Confidence

Having made these observations that the effects of 'ideal' evidence are not always of the kind one might naively expect, we now focus on *improvement in confidence* as our stricter definition of a 'supportive argument' (as opposed to mere 'ideal evidence': $V = \text{verified}$ and/or $T = \text{no failures}$). Thus, we will call an argument *supportive if it improves upon the prior confidence*. So the testing leg with outcome T is called 'supportive' whenever $P(S \leq s | T) > P(S \leq s)$. Similarly a verification leg with outcome V is called 'supportive' whenever $P(S \leq s | V) > P(S \leq s)$. The dual-legged argument is 'supportive' when $P(S \leq s | VT) > P(S \leq s)$.

For each set of model parameters, there are actually four different cases for which one can ask whether a confidence improvement occurs on receipt of ideal evidence from an argument leg. A system assessor may obtain either *testing-leg* evidence $T = \text{no failures}$, or formal *verification-leg* evidence $V = \text{verified}$, in each case with or without evidence of the other kind being already present. We explained in §6.1 that, as our model stands, all these four questions may be framed mathematically as the questions as to whether a single expression, which can be either of (18) or (20), increases or decreases when ideal evidence is received. In terms of this expression, ideal testing evidence is added by changing from $n=0$ to a general non-zero n value (and making the associated change to μ and μ' , from 1 to the corresponding values). Similarly ideal verification evidence is added by changing from $(\alpha, \xi) = (0, 1)$ to the general pair.

Thinking of the values of our doubt expression (18) as laid out as a square with each side corresponding to the receipt of "ideal evidence" V or T , we can depict the conditions described in the questions above diagrammatically. In our diagrams, a downward arrow corresponds to the receipt of ideal testing leg evidence and a rightward arrow corresponds to the receipt of ideal verification leg evidence. So the top, left vertex corresponds to prior confidence and the bottom right to the final confidence after evidence from both legs is in. Following an arrow represents 'progress', in the sense that further ideal evidence is received. The ' $>$ ' (beside a right-pointing arrow) and ' \vee ' (beside a downward-pointing arrow) symbols are used to indicate a decrease in confi-

dence as we move along the arrow. Thus the counter-intuitive situations which we have called non-*supportive* single legs look like

$$\begin{array}{ccc}
 00 & \xrightarrow{>} & 0V \\
 \downarrow & & \downarrow \\
 T0 & \longrightarrow & TV
 \end{array} \tag{27}$$

for verification leg evidence, and like

$$\begin{array}{ccc}
 00 & \longrightarrow & 0V \\
 \downarrow^{\vee} & & \downarrow \\
 T0 & \longrightarrow & TV
 \end{array} \tag{28}$$

for testing evidence. In these diagrams a *zero* represents the *absence of evidence* for one or other leg.

We are now in a position to ask whether counter-intuitive results are possible within our model for two-legged arguments. It is worth recalling here the *systems* metaphor that underpins the intuition behind the use of such arguments. It is well-known that the reliability of a 1-out-of-2 system will always be greater than or equal to the reliability of the best of the two components. Is this true of our two-legged arguments: in particular, is it always the case that confidence will increase if we add a second *supportive* argument leg to an initial supportive leg? That is, can either of the following two cases occur:

$$\begin{array}{ccc}
 00 & \xrightarrow{<} & 0V \\
 \downarrow^{\wedge} & & \downarrow \\
 T0 & \xrightarrow{>} & TV
 \end{array} \tag{29}$$

or

$$\begin{array}{ccc}
 00 & \xrightarrow{<} & 0V \\
 \downarrow^{\wedge} & & \downarrow^{\vee} \\
 T0 & \longrightarrow & TV
 \end{array} \tag{30}$$

We have examined this question numerically and tentatively conclude that, for our current model, the answers are: no, the scenario depicted in (30) is ruled

out; but yes it *is possible* for the case of (29) to occur, in which a supportive verification leg can *depress* confidence when it is added to pre-existing ideal testing evidence, which itself is supportive when considered alone.

For a numerical example of this case (29), consider the two-legged argument with evidence $s=0.001$, $n=10,006$, $a=0.0807$, $b=0.0192$, $a'=8.2408 \times 10^{-3}$, $b'=0.044813$, $\alpha=0.12419$, $\xi=4.9315 \times 10^{-6}$, $p_{oi}=6.91181 \times 10^{-3}$, $p_{oc}=9.69767 \times 10^{-3}$,

$$\pi^i = \begin{bmatrix} 0.99965 & 5.50587 \times 10^{-6} \\ 1.19185 \times 10^{-5} & 3.28401 \times 10^{-4} \end{bmatrix}.$$

With these parameters, the prior confidence 0.8001 is considerably improved, to 0.9659, by a positive system verification outcome. *Without* any such verification leg, but with instead ideal testing evidence from the 10,006 trials, confidence improves to as much as 0.999627. However, when the positive verification evidence is added, as one of two legs, to this ideal testing outcome, then much of this large confidence gain from the testing leg alone is *lost*, with a confidence now of only 0.9671, which – while still an improvement over the prior confidence – is not *as great* an improvement as was obtained from the testing leg alone. I.e., the act of *adding* a supportive verification leg *to* a testing leg has actually *lowered* our confidence in the claim. When comparing the confidence resulting from a two-legged argument against the prior confidence and against that from each single leg individually, we can lay out these four confidence values in a 2×2 matrix format

$$P(S \leq 10^{-3} | obs.) = \left(\begin{array}{cc} P(S \leq 10^{-3}) & P(S \leq 10^{-3} | V) \\ P(S \leq 10^{-3} | T) & P(S \leq 10^{-3} | VT) \end{array} \right) = \left(\begin{array}{cc} 0.8001 & 0.9659 \\ 0.999627 & 0.9671 \end{array} \right) \quad (31)$$

matching the layout of our square diagrams above. We have used curved brackets around the matrix to distinguish it visually from our other frequently used 2×2 -matrix notation for joint distributions of variables ZO (for which we will reserve square-bracketed matrix notation). Using the same layout as that in (27) to (30) – with the prior beliefs as the top left hand square-bracketed matrix, and the beliefs emanating from a two-legged argument at the bottom right, etc – the four possible sets of beliefs about the ZO pair for this example,

under the four possible conditions of evidence discussed, appear as

$$\begin{aligned}
P(ZO | obs.) &= \begin{pmatrix} P(ZO) & P(ZO|V) \\ P(ZO|T) & P(ZO|VT) \end{pmatrix} \\
&= \begin{pmatrix} \begin{bmatrix} 0.999654 & 5.5059 \times 10^{-6} \\ 1.1919 \times 10^{-5} & 3.2840 \times 10^{-4} \end{bmatrix} & \begin{bmatrix} 0.96148 & 5.2956 \times 10^{-6} \\ 1.3489 \times 10^{-3} & 0.037168 \end{bmatrix} \\ \begin{bmatrix} 0.999572 & 7.0415 \times 10^{-6} \\ 1.4354 \times 10^{-6} & 4.2000 \times 10^{-4} \end{bmatrix} & \begin{bmatrix} 0.96264 & 5.3027 \times 10^{-6} \\ 1.2720 \times 10^{-4} & 0.037218 \end{bmatrix} \end{pmatrix} \tag{32}
\end{aligned}$$

Some further insight into the apparently counter-intuitive result – that adding a supportive verification leg to the testing leg can reduce confidence in the dependability claim – comes from examining these intermediate numerical results. Consider first the prior belief represented by the top-left matrix in (32): in particular note that the leading diagonal of this matrix suggests that the *a priori* beliefs about Z and O are positively associated. That is, belief that the specification is incorrect results in a stronger belief the oracle is incorrect, and vice-versa.

Seeing 10,006 test cases executed without failure, in the first argument leg, results in increased doubt about the correctness of the oracle: from $5.5059 \times 10^{-6} + 3.2840 \times 10^{-4} = 3.3391 \times 10^{-4}$ *a priori* to $7.0415 \times 10^{-6} + 4.2000 \times 10^{-4} = 4.2704 \times 10^{-4}$ after the test data is seen. Because of the association between beliefs about Z and O , this supportive evidence from testing undermines confidence in the specification correctness: doubt increases from $1.1919 \times 10^{-5} + 3.2840 \times 10^{-4} = 3.4032 \times 10^{-4}$ *a priori*, to $1.4354 \times 10^{-6} + 4.2000 \times 10^{-4} = 4.2144 \times 10^{-4}$ after seeing the testing evidence.

The verification leg is supportive when we have no testing evidence (i.e. it increases confidence from its *a priori* value). But the above reasoning shows that the presence of (successful) testing can undermine the contribution that the verification leg makes to overall confidence in the dependability claim when both argument legs are present. In fact this undermining can be so severe that adding the verification leg makes things worse, compared with having only the testing leg.

Examination of the parameter values used to construct this example might cause one to conclude that some of them seem unlikely to be realistic beliefs of experts about real systems. E.g. consider the virtual prior certainty, according to these parameters, that the specification is correct; or the asymptotes ($b, b' < 1$) of the two beta distributions at the 1 end of the unit interval $[0, 1]$. It remains

to determine whether actual realistic beliefs could also exhibit property (29).

Numerically we have been unable to obtain a similar example with T and V interchanged, i.e. satisfying (30). We conjecture that such an example may prove to be analytically impossible for our current model, while we impose our stochastic ordering constraint (10). Although we have not obtained experimentally the reversal of ordering occurring on the right hand side of (30), one finding that is almost as surprising is that we can find examples in which – in our terminology explained earlier – a significantly supportive testing leg, when added to a significantly supportive verification leg creates only a negligibly small improvement in confidence of the two-legged argument over the confidence obtained from the verification leg alone. A numerical example of this is provided by the values $s=0.001$, $n=19,921$, $a=0.092728$, $b=2.4768$, $a'=0.13423$, $b'=3.8705$, $\alpha=9.8691 \times 10^{-3}$, $\xi=2.8029 \times 10^{-7}$, $p_{0i}=1.39760 \times 10^{-3}$, $p_{0c}=0.18737$

$$\pi = \begin{bmatrix} 0.47491 & 0.09055 \\ 1.80783 \times 10^{-4} & 0.43436 \end{bmatrix}.$$

With these values, the ideal evidence produces three confidence levels, laid out next to the prior confidence as in the last example

$$P(S \leq 10^{-3} | obs.) = \begin{pmatrix} 0.59125 & 0.67018 \\ 0.70759 & 0.67025 \end{pmatrix}. \quad (33)$$

Here, the supportive testing evidence produces an improvement of the two-legged over the verification-only argument which is less than one in the fourth-significant decimal digit.

8 Special Case of Claims for Perfection, $S=0$

If, instead of a claimed upper bound $S \leq s$ on pfd, we make a claim for perfection, i.e. $S=0$, then we obtain a special case of the above expressions for confidence, doubt for which certain of the counter-intuitive results demonstrated above become no longer possible. This substitution corresponds to the special case where our confidence refers to a claim that the system is *perfectly reliable*, rather than merely that its pfd *does not exceed some positive threshold* value $S \leq s > 0$.

Firstly, we can show that for such a perfection claim, operation that is completely failure-free throughout testing *always* constitutes a *supportive* argument leg in the sense we identified in §7.1, for the simple reason that our model assumptions clearly make $S=0$ and $S=0 \& T$ identical events. Thus, we

must have a confidence, from the testing leg only given by

$$P(S=0|T) = \frac{P(S=0 \& T)}{P(T)} = \frac{P(S=0)}{P(T)} \geq P(S=0), \quad (34)$$

that is no less than the prior confidence. Further, for this perfection claim we can prove easily the special case of the result that, for the more general claim, we have been so far able to verify only numerically without an analytic proof: ideal testing evidence added to a positive verification outcome to produce a two-legged argument *always*¹⁷ improves upon the confidence emanating from the verification leg alone. Essentially the same short proof as that given above works again here. Our model assumptions make $S=0 \& V$ and $S=0 \& VT$ identical events. So we have a final confidence from the two-legged argument

$$P(S=0|VT) = \frac{P(S=0 \& VT)}{P(VT)} = \frac{P(S=0 \& V)}{P(VT)} \geq \frac{P(S=0 \& V)}{P(V)} = P(S=0|V), \quad (35)$$

the confidence in this perfection claim emanating from the single verification argument.

9 Summary and Conclusions

As we have said earlier in this paper, the BBN we have studied here has been an over-simplified one. The first simplification is in only taking account of statistical testing and proof evidence: we have ignored other kinds of evidence that would clearly be relevant in practice – for example evidence concerning the quality and competence of the personnel involved at all stages. Furthermore, each of the argument legs considered here is itself unrealistically simplified: e.g. the testing leg ignores important issues concerning the accuracy of the operational profile. We have also artificially reduced some of the state spaces to Boolean. Finally, to make the mathematics tractable, we have had to introduce some simplifying assumptions that (rather tentatively) we claim to be ‘conservative’.

Our main reason for these simplifications lay in our desire to carry out the analysis completely analytically. We wanted to obtain complete analytic expressions for posterior distributions in terms of parametric families of input node probability distributions. This contrasts with the more common approach to BBN analysis in which numerical expressions – e.g. involving elicited expert beliefs – are manipulated using tools like Hugin [22]. Simply populating the

¹⁷ even without a restriction that the verification leg must be *supportive*

node probability tables in this way results only in a single numerical posterior distribution for the goal variable S.

Our intention here was to obtain greater insight into the factors that determine the efficacy of arguments, and in particular of multi-legged arguments. We started from the position that efficacy would be judged by the confidence that the arguments engendered in dependability claims. We wanted a better understanding of how confidence is determined by factors such as the doubt in the truth of assumptions underpinning the arguments. In particular, we sought to understand the importance of association – e.g. between the doubts associated with assumptions for different argument legs – in determining the effectiveness of the multi-legged argument approach.

In spite of the great simplification we have applied, the model turns out to be quite complex and difficult to understand. We were somewhat surprised by this, and we regard it as a strong warning against a naive trust in the results of a conventional numerical analysis of a BBN like this. In particular, we believe that the analytic approach has exposed some non-intuitive and surprising results that would not be noticed in a conventional numerical analysis. It would be possible for one to be lulled into a false sense of certainty and security, and believe numerical consequences that one would not believe with the benefit of greater insight (e.g. offered by the kind of analysis we have conducted).

These remarks confirm our long-held view that BBNs need to be treated with great respect and humility [23,24]. The Bayesian approach is clearly the right one for the representation of uncertainty, and the BBN formalism has immeasurably aided understanding and construction of complex probability models. But the very seductiveness of the approach – particularly in its automated numerical form – can bring unwarranted confidence in the results.

When we set out on this work, we had in mind an analogy with the use of diversity in systems – multiple diverse channels – to increase reliability and safety. It seemed plausible that multi-legged arguments could be used similarly to increase confidence in claims about dependability (e.g. safety). It turns out that this analogy breaks down in surprising ways.

One way in which the systems metaphor breaks down concerns composability of arguments. Our example illustrates that it can be straightforward to *decompose* a model for a two-legged argument, producing two derived, single-leg models as special cases corresponding to a degenerate observation for one or other argument leg. However there is no standard reverse operation of *composition* starting from a given pair of single-leg models. The representation of *dependence* via variables which link the two argument legs is an additional and difficult modelling task, whose solution is integral to any meaningful model of a two-legged argument.

Another surprising way in which the systems/arguments analogy breaks down concerns efficacy: whereas it is easy to show that, for systems, a diverse 1-out-of-2 system is always more reliable than each of its component channels, the same is not true of arguments. We cannot be sure that the confidence in a dependability claim arising from a diverse 2-legged ('1-out-of-2') argument is greater than that arising from either of the single argument legs. Indeed, we have examples where a single leg is to be preferred to the same leg aided by a further supportive argument leg: i.e. additional 'good news' is not necessarily beneficial and can even be detrimental.

Such results are, at first glance, counter-intuitive: indeed, if they had been obtained from a purely numerical analysis they would be hard to explain. The more detailed analysis here has the advantage of showing how this kind of thing can happen, by revealing the subtle interplay between assumptions and evidence, both within and between legs. It thus provides warnings against drawing simplistic – albeit intuitively plausible – conclusions. It cannot be too strongly emphasised that it is only through the completely analytical treatment – difficult though it is – that we get these insights.

On the other hand, it is clear that in many cases – as might be expected – multi-legged arguments *do* bring benefits in terms of increased confidence in dependability claims compared with single arguments, as we have shown in Section 6. From a practical point of view, we would like to know exactly when such benefits can be expected, and how extensive they might be. Ideally, we would like to be able to design multi-legged arguments – before the expensive process of evidence-collection begins – so that confidence in a dependability claim will be gained *most cost-effectively*.

We do not claim that our work here enables this to be done. But we do believe that it is a useful beginning in understanding some of the key issues. Thus, for example, in Section 6 we show how it is possible with our analytical treatment of the BBN to perform 'what if' calculations on the effect of dependence, in assumption doubts for the two legs, upon the confidence that the two-legged argument provides in the claim. It is easy to see how this kind of study could be used to compare different possible multi-legged arguments before committing to the expense of deploying them in a particular dependability case.

Concerning the general efficacy of diverse arguments in real-life applications, it is clear that more work is needed. It would be interesting to know, for example, whether the kinds of parameters that are realistic (e.g. for experts' beliefs) in our simplified model result in 2-legged arguments that are almost always effective, in the sense of being better than the constituent legs. Are the exceptions that we have identified in some sense 'not believable' when real experts assess real systems? To what extent are some of these results the consequence of our need to make 'conservative' assumptions for mathemati-

cal tractability? The results presented here are largely neutral on such issues: they concern what *might* happen, rather than what *will* happen when real experts assess real systems. It is worth stating, however, that when we constructed the simplified example used in the paper, we did not anticipate those consequences that we have called ‘counter-intuitive’: it seems possible, even after considerable reflection, to be surprised by what is implied by a complex model. This is likely to be true *a fortiori* for more realistic, and thus more complex, models.

We end by acknowledging that this kind of work assumes the need for a higher level of formality than is presently the case. A formal treatment of argument efficacy in terms of (claim, confidence) pairs is rare even in dependability cases that support decisions about safety-critical systems. It might be asked why we cannot hide all the complexity of analyses like the ones here, and simply rely on (say) the judgement of an experienced expert that a system is safe to deploy. One of the most forceful lessons we have learned from this work is that *the devil lies in the details*. The complexity involved here seems to be inherent, and you ignore it at your peril. Thus if, as we claim, a purely numerical BBN can get things wrong (by ignoring some of the complexity), we must be concerned that an even more informal qualitative approach can get things wrong, since it ignores even more of the complexity.

Acknowledgement

This work was partially supported by the DIRC project (‘Interdisciplinary Research Collaboration in Dependability of Computer-Based Systems’) funded by UK Engineering and Physical Sciences Research Council (EPSRC), and by the DISPO (DIverse Software PRoject) projects, funded by British Energy Generation Ltd and BNFL Magnox Generation under the Nuclear Research Programme via contracts PP/40030532 and PP/96523/MB.

References

- [1] Ministry of Defence, Requirements for safety related software in defence equipment, UK Defence Standard Def-Stan 00-55, issue 2 (August 1997).
- [2] Civil Aviation Authority, Regulatory objective for software safety assurance in air traffic service equipment, CAA SW01 (2001).
- [3] R. E. Bloomfield, B. Littlewood, Multi-legged arguments: The impact of diversity upon confidence in dependability arguments, in: Proceedings DSN 2003, IEEE Computer Society, 2003, pp. 25–34.

- [4] D. E. Eckhardt, L. D. Lee, A theoretical basis for the analysis of multi-version software subject to coincident errors, *IEEE Transactions on Software Engineering* 11 (1985) 1511–17.
- [5] B. Littlewood, D. R. Miller, A conceptual model of multi-version software, *Digest of 17th Fault Tolerant Computing Symposium* (1987) 150–55.
- [6] J. C. Knight, N. G. Leveson, An experimental evaluation of the assumption of independence in multi-version programming, *IEEE Trans. on Soft. Eng.* 12 (1986) 96–109.
- [7] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Mathematics and Its Applications, Morgan Kaufmann, San Mateo, California, 1988, revised 2nd printing 1991.
- [8] M. Volf, M. Studený, A graphical characterisation of the largest chain graphs, *International Journal of Approximate Reasoning* 20 (3) (1999) 209–36, <ftp://ftp.utia.cas.cz/pub/staff/studeny/volstu.ps>.
- [9] D. R. Wright, Elicitation and validation of graphical dependability models, Tech. rep., City University, ROPA Project Report: www.csr.city.ac.uk/people/david.wright/ropa/ (2003).
- [10] A. P. Dawid, Conditional independence in statistical theory, *Journal Royal Statistical Society, Series B* 41 (1) (1979) 1–31, with discussion.
- [11] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, D. J. Spiegelhalter, *Probabilistic Networks and Expert Systems*, Statistics for Engineering and Information Science, Springer-Verlag, New York, 1999.
- [12] S. L. Lauritzen, *Graphical Models*, Oxford Statistical Science Series, Clarendon Press, Oxford, 1996.
- [13] G. Shafer, *Probabilistic Expert Systems*, CBMS-NSF Regional Conf. Ser. in Applied Math., Society for Industrial & Applied Mathematics, Philadelphia, 1996.
- [14] A. P. Dawid, Conditional independence for statistical operations, *Annals of Statistics* 8 (3) (1980) 598–617.
- [15] M. Studený, On mathematical description of probabilistic conditional independence structures, Dr. of Science Thesis, Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Prague (May 2001).
- [16] N. Wermuth, S. L. Lauritzen, On substantive research hypotheses, conditional independence graphs and graphical chain models, *Journal Royal Statistical Society, Series B* 52 (1) (1990) 21–72, with discussion.
- [17] D. Wright, Elicitation and validation of graphical dependability models, in: S. Anderson, M. Felici, B. Littlewood (Eds.), *SAFECOMP 2003*, Edinburgh, UK, 23-6 Sept., Springer-Verlag Heidelberg, 2003, pp. 8–21.

- [18] G. de Barra, Measure Theory and Integration, Mathematics and Its Applications, Ellis Horwood, Chichester, 1981.
- [19] M. H. DeGroot, Optimal Statistical Decisions, Wiley, 2004, wiley Classics Library Edition. ISBN 0-471-68029-X.
- [20] M. Abramowitz, I. A. Stegun (Eds.), Handbook of Mathematical Functions, Dover, New York, 1970, <http://www.math.sfu.ca/~cbm/aands/>.
- [21] B. Littlewood, D. Wright, Some conservative stopping rules for the operational testing of safety-critical software, IEEE Transactions on Software Engineering 23 (11) (1997) 673–83.
- [22] S. K. Andersen, K. G. Olesen, F. V. Jensen, F. Jensen, Hugin—a shell for building Bayesian belief universes for expert systems, in: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, 1989, pp. 1080–84.
- [23] S. Yih, C.-F. Fan, Search for the unnecessary, Nuclear Engineering International (2001) 24–6.
- [24] R. Bloomfield, P.-J. Courtois, L. Strigini, B. Littlewood, Letter to the editor, Nuclear Engineering International (2002) 11.

Table 1: How the Conditional Probability Tables Combine to Produce $P(S \& ideal \text{ obs.})$

S	0				> 0			
	correct		incorrect		correct		incorrect	
Z	correct	incorrect	correct	incorrect	correct	incorrect	correct	incorrect
O								
$P(T=no \text{ failures} OS)$	1	1	1	1	$(1-S)^n$	1	$(1-S)^n$	1
$P(S Z)$	$P(S=0 Z=corr.)$		$P(S=0 Z=incorr.)$		$P(S Z=corr.)$ (a pdf)		$P(S Z=incorr.)$ (a pdf)	
$P(V=verified SZ)$	$1-\alpha$	$1-\alpha$	1	1	ξ	ξ	1	1
$P(S \& ideal \text{ obs.})$	$(1-\alpha)P(S=0 Z=corr.)P(Z=corr.) + P(S=0 Z=incorr.)P(Z=incorr.)$		$\xi P(S Z=corr.) [(1-S)^n P(ZO=(corr.,corr.)) + P(ZO=(incorr.,incorr.))]$		$+ P(S Z=incorr.) [(1-S)^n P(ZO=(incorr.,corr.)) + P(ZO=(incorr.,incorr.))]$			
$P(ideal \text{ obs.})$	$(1-\alpha)P(S=0 Z=corr.)P(Z=corr.) + P(S=0 Z=incorr.)P(Z=incorr.) +$		$\int_{0^+ < S \leq 1} \xi P(S Z=corr.) [(1-S)^n P(ZO=(corr.,corr.)) + P(ZO=(incorr.,corr.))] + P(S Z=incorr.) [(1-S)^n P(ZO=(incorr.,incorr.))] dS$					

(* understanding that this integral specifically excludes any probability masses at $S=0$ of the conditional distributions of S given Z .)

A general modeling approach and its application to a UMTS network with soft-handover mechanism

- TECHNICAL REPORT RCL060501 -
(UNIVERSITY OF FIRENZE, DIP. SISTEMI E INFORMATICA)
(MAY, 2006)

Paolo Lollini
Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze
viale Morgagni 65,
50134, Firenze, Italy
email: lollini@unifi.it

Andrea Bondavalli
Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze
viale Morgagni 65,
50134, Firenze, Italy
email: bondavalli@unifi.it

Felicità Di Giandomenico
ISTI Institute
Italian National Research Council
via Moruzzi 1,
56124, Pisa, Italy
email: digiandomenico@isti.cnr.it

ABSTRACT

This paper presents a general modular modeling approach applicable to the wide class of cellular systems, including GSM, GPRS and UMTS networks. It is based on the identification of the building-blocks, the basic parts of the system to be modeled, and of their interfaces, the part of the building-block models that can interact with the others, and it enhances the modularity, reusability, scalability and the maintenance of the overall model. In the second part of this work we apply the modeling approach to a concrete case-study, an UMTS network with overlapping cells, also accounting for the soft-handover mechanism. The goal is to analyze the QoS perceived by the users camped in the normal operational mode and during outage events that decrease the availability of the network resources.

KEY WORDS

Modeling Framework - Cellular Network - UMTS - Stochastic Activity Network - Soft Handover - QoS Analysis.

1 Introduction

A cellular network is a radio network made up of a number of possibly overlapping radio cells (or just cells) each served by a fixed transmitter (called cell site or base station). These cells are used to cover different areas in order to provide radio coverage over an area wider than the area of one cell. The most common example of a cellular networks are mobile phone networks, like GSM (Global System for Mobile Communications), GPRS (General Packet Radio System) and UMTS (Universal Mobile Telecommunications System). These networks have very different physical and functional characteristics.

GSM [1] uses a frequency division multiple access (FDMA) technology. It handles voice traffic requirements of the mobile communication by providing a circuit switched mode of operation (high-speed circuit switched data). Circuit switching provides the customer with a dedicated channel all the way to the destination. The customer has exclusive use of the circuit for the duration of the call, and is charged for the duration of the call.

GPRS [1] provides packet radio access (packet switching) for mobile GSM and time-division multiple access (TDMA) users. With packet switching, the operator assigns one or more dedicated channels specifically for shared use. These channels are up and running 24 hours a day, and when you need to transfer data, you access a channel and transmit your data. Packet switching is more efficient than circuit switching.

UMTS [2, 3] is a third generation (3G) mobile communications system that provides a range of broadband services to the world of wireless and mobile communications. It preserves the global roaming capability of second generation GSM/GPRS networks and provides new enhanced capabilities. The UMTS takes a phased approach toward an all-IP network by extending second generation (2G) GSM/GPRS networks and using Wide-band Code Division Multiple Access (CDMA) technology. Handover capability between the UMTS and GSM is supported. The GPRS is the convergence point between the 2G technologies and the packet-switched domain of the 3G UMTS.

As we can note, the mobile networks (as well as cellular networks) are typically characterized by different technologies, different architectures, different interfaces, different protocols, different access modes. These distinctive characteristics constitute a very hard problem when such type of systems have to be analyzed in order to evaluate some specific indicators, like availability, reliability or, in general, QoS measures. Model-based analysis is typically employed to this purpose, however the modeling and solution process is always profiled considering a specific mobile phone network technology and a specific network topology, and then the effort produced to analyze a particular network can be hardly reused in other contexts, just considering, for example, the same network technology with a different topology.

In this paper we give a contribution towards the definition of a general modeling framework that is not restricted to the analysis of a particular class of mobile phone network. Inspired by [4], the main modeling components are identified with respect to the functions they perform, without detailing how these models are actually built or the used modeling formalism. Such models can interact each other through some “model interfaces” and a general compositional operator. The obtained modeling structure enhances the modularity, reusability, scalability and the maintenance of the overall model.

The rest of this paper is organized as follows. Section 2 presents the modular modeling framework specifically tailored for the cellular systems. In Section 3 the modeling approach is applied to a specific case-study concerning a UMTS network with soft-handover mechanisms, also defining the measures of interest and the adopted modeling assumptions. The models built using SAN formalism are presented in Section 4, and some numerical results are presented in Section 5. Conclusions are drawn in Section 6.

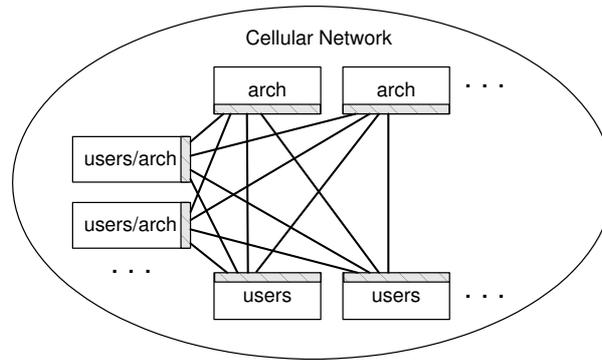


Figure 1. Building-blocks and their interactions

2 The modeling framework

The principle of the modular construction/composition approach is to build complex models in a modular way through a composition of its submodels. This principle has been extensively applied to model very different systems, like multipurpose multiprocessor systems ([5]) and railway interlocking systems ([6]).

This general principle is here instantiated in the context of cellular systems. We identify some basic models, the *building-blocks*, each one representing a well-specified part of the system. Such basic models interact each other through the model *interfaces*, that are the part of the building-blocks that can interact with the other. The interaction happens through the application of a sort of general compositional operator that enables the “communications” among different models.

In order to identify the building-blocks, we follow a top-down approach. A cellular network can be seen as a set of partially overlapping cells. Each cell can be described in terms of architecture and users. The architecture of a cell depends on the technology we are considering and it could be further decomposed in several simpler elements, like Base Stations, Core Network, UMTS Radio Access Network, and so on. The users are the network consumer: they require services that can demand different traffic workload level, like video streaming, phone call, and so on. With respect to these considerations, we identify the following high-level building-blocks for a cellular network:

The “arch” model. This model describes the architecture of the cell with respect to the purpose of the analysis. For example, recent works dealing with GPRS network [7, 8] focus on the Random Access Procedure, that is a method to access to the network that may cause collisions among requests by different User Equipments (UE), thus worsening the expected Quality of Service (QoS).

The “users” model. This model represents the behavior of a class of users characterized by the same type of requested service (e.g. video streaming), service time, inter-request time, and so on. Therefore, the users belonging to the same class are undistinguishable.

The “users/arch” model. It enables the communication between the users and the cell (or the cells) in which they are connected to, that mainly consists in the definition of the network topology and in the allocation or deallocation of the traffic channels (if available) to satisfy the service requests.

In Figure 1 we show the view of the cellular system as composition of interacting building-blocks. The shaped model areas are the interfaces, that are the parts of the models that are directly connected to the others. Note that at this level of abstraction a building-block belonging to a class can interact with all the others. For example, the user/arch model can interact both with more than one cell (e.g. when a user is served by more cells - Soft Handover) and with more than one class of users (e.g. when different types of users are served by the same cell).

At this level of abstraction we can not explicitly describe what is an interaction, since its definition strictly depends on the particular modeling formalism used to build the models. In this context the line connecting two models can be seen as a compositional operator that enables the interactions between the connected models through the respective interfaces.

The overall model of Figure 1 has very important characteristics:

It is built in a modular way through the composition of simpler submodels. This modular approach helps to cope with the possibly too high complexity which would be incurred in when building the model of a system as a whole.

Each model captures the behavior of a specific system function, thus improving the “readability” of the overall model.

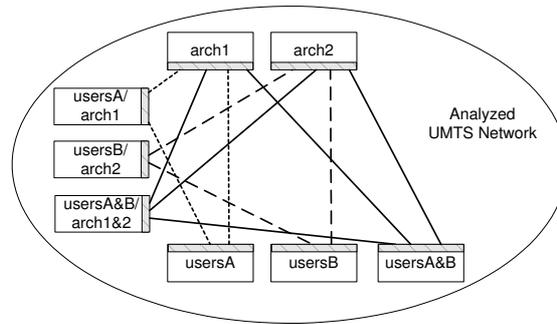


Figure 2. UMTS network model with soft-handover

The whole model is easily modifiable, since the parts of the model that are not interfaces can be modified without impacting on the other models.

The model is reusable, in the sense that parts of the models could be reused in the modeling of other different cellular networks. This property is extremely useful when, for example, we modify the network topology of an already modeled cellular system. In this case the overall model can be reused just modifying the interface of the “users/arch” model.

3 Case-study: QoS analysis of a UMTS network with soft-handover mechanism

In this section we apply the proposed modeling approach to represent the behavior of a UMTS network composed by two partially overlapping cells and accounting for the soft-handover mechanism ([9]).

A UMTS network consists of a set of local areas, each one composed by a number of overlapping cells. Two cells are overlapping if they share radio frequencies and users (mobile equipments). Each cell is characterized by the following building blocks (see Figure 2):

the “arch1” and “arch2” models, that implement the resource allocation;

the “usersA” and “usersB” models, that represent the users that are connected to one cell only. They are characterized by the same service request frequency, probability to obtain a dedicated traffic channel and the same service completion time;

the “usersA&B” model is the class of users that can be connected simultaneously to more than one cell using the soft-handover mechanism, since they are in the overlapping area between the two cells. It describes the service request frequency, the probability to obtain a dedicated traffic channel and the service completion time. Moreover, they are characterized by the same service request frequency, the same probability to obtain a dedicated traffic channel and the same service completion time;

the “usersA/arch1”, “usersB/arch2” and “usersA&B/arch1&2” models describe the interactions among the users and the corresponding architecture models.

3.1 Measures of interest

The focus is on the effects of outages on the QoS perceived by the users camped in the network. An outage results in an unavailability of one cell; as a consequence, some connections are lost and some other connections experiment a QoS degradation since the users in soft-handover now issue the service request to one cell only. In this context, we analyze several measures, among them: i) the number of allocated traffic channels for each user class, and ii) the probability that a service request can not be satisfied for each user class.

3.2 Assumptions

The assumptions we introduced in the modeling phase are here sketched:

the network is composed of two overlapping cells, and the users are uniformly distributed inside each cell;

the cells can be active (all the resources are available), down (no available resources) or partially active (to represent partial outages of some components);

we consider an admission control algorithm based on the workload of the UMTS cell: a new call is accepted if the workload level reached after adding the call does not exceed a pre-specified threshold, both in uplink and in downlink;

the number of users camped in each cell is constant, and no user can move from a service class to another;

all the users can be distinguished based on the type of service they require, and they all have the same priority;

service classes differ for the throughput, the traffic workload they induce on the cell and the service time;

if the cell accepts the service request, a Dedicated CHannel (DCH) is assigned to the user until the service is completed.

the outage occurs following a deterministic distribution and, when it occurs, the affected cell goes down. Therefore, the connections of the users camped in the downed cell are immediately lost, and the users in soft-handover increase their transmission power to connect to the remaining active cell;

the cell can not be repaired;

when the outage occurs, both cells are working in steady-state condition;

4 Model implementation

All the models have been built using Stochastic Activity Networks (SANs) formalism ([10]), that is a generalization of SPNs and have some similarities to the GSPN formalism.

Following the object oriented philosophy, we develop a sort of “template” models, one for each building block previously identified. The overall UMTS model results from the composition of some “instances” of such classes. The submodels will be composed together using the Join operator ([11]). It is a composition technique for SAN that combines models by sharing state, thus decreasing the overall number of states of the entire model. The Join operator takes as input a) a set of submodels and b) some shared places owning to different submodels of the set, and provides as output a new model that comprehends all the joined submodels’ elements (places, arcs, activities) but with the shared places merged in a unique one. Therefore, the shared places represent the “interfaces” among models.

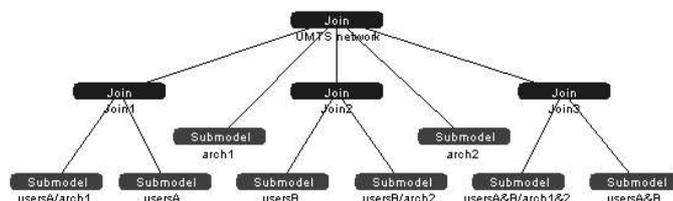


Figure 3. Overall UMTS network model

In Figure 3 we depict the composed model that represents the behavior of a UMTS network with two partially overlapping cells. There are two models representing the cell architectures (arch1 and arch2), three models representing the class of users camped in the network (two for the users camped in one cell only - usersA and usersB - and one for the users camped in the overlapping area between the two cells - usersA&B), and three models describing the corresponding users/arch interactions (usersA/arch1, usersB/arch2, and usersA&B/arch1&2). The places shared among models are identified in the following Subsections in which we sketch the structure of each model.

4.1 “arch” model

The model presented in Figure 4 represents the architecture of a UMTS cell (it is a template model, in the sense that it has to be instantiated to represent the architectural behavior of the two different cells considered in our network segment).

The left part of the model represents the variations of the state of activity of the cell. During normal conditions, the cell is properly working (place *Work* contains one token). When an outage occurs (transition *T_{Work}* fires), the cell enters in one of the possible states *State1*, *State2*, ..., *Down*. Place *Down* represents the complete unavailability of the cell, and it is

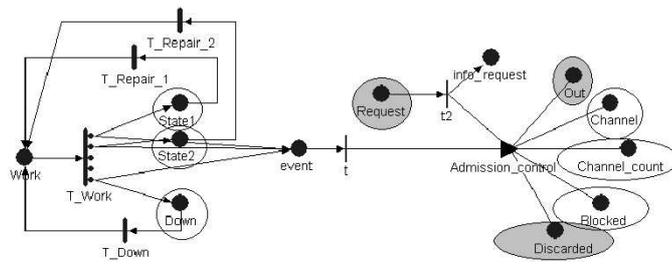


Figure 4. The architectural model

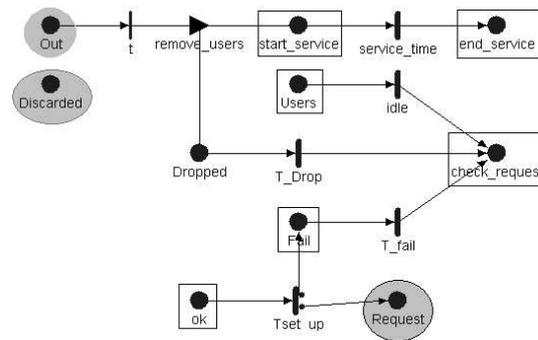


Figure 5. The model for the class of generic users

the case of interest. Each degraded state can be repaired after a time defined in the deterministic transitions T_{repair_1} , T_{repair_2} , ..., T_{Down} .

The output gate *Admission_control* executes the admission control algorithm. At each instant of time, it checks the cell traffic workload (both in uplink and in downlink) and the number of allocated traffic channels. When a new service request is issued from a user ($mark(Request) > 0$), the admission control algorithm verifies that the new traffic workload level remains below a certain threshold level and, in this case, it accepts the new service request. In this case, one token is added to place *Channel*, representing the availability of a traffic channel, and one token is added to place *Channel_count*, that counts the number of allocated traffic channels. If the new service request can not be accepted (the admission control algorithm fails), then one token is added to place *Blocked*. The output gate *Admission_control* is also in charge of computing the number of services to be blocked in order to move the workload level under a pre-specified threshold, for example in case of total outage. In this case, an appropriate number of tokens (*users*) is removed from place *Channel_count* and added to the place *Discarded*.

For the sake of readability, in Figure 4 we depict with a shaped circle the places shared with the user model, and with a non-shaped circle the places shared with the users/arch model.

4.2 “usersA” model

In Figure 5 we depict the “userA” model, that is the model representing the behavior of the class of users that are connected to one cell only.

Place *Users* represents the idle users, that are the users that are not requiring any service to the network. The inter-request time is defined by the exponential transition *idle*, that at firing time moves a token in *check_request* that represents the users that have sent a service request to the network. If the cell is up, the service request arrives in place *ok* and then the random access channel (RACH) procedure is performed in order to obtain a slot for issuing the service request. Transition *Tset up* defines the duration of the RACH procedure. If it succeeds (with a given probability), one token is added to place *Request*, otherwise the token is added to place *Fail*. Tokens in place *Fail* represent the users whose service request has not been accepted because of a failure of the admission control algorithm or of the RACH procedure. Such unsatisfied users issue a new service request after an exponential time defined in the transition *T_fail*. When a traffic channel has been allocated to a user, a token arrives in place *start_service* (the user is being served) and, after a service time defined by the exponential transition *service_time*, the service request is completed and then the token is moved to place *end_service*. A token in place *Out* enables the instantaneous transition *t*, the output function defined in the output gate *remove_users* is executed

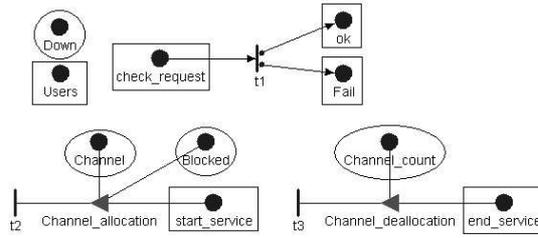


Figure 6. The model for the interactions among users and architectures

Parameters	Meanings	Values
Δ_{dl}	Downlink workload increment per service	0.013 db
Δ_{ul}	Uplink workload increment per service	0.016 db
SH_users	Users in soft-handover	16
Gen_users_A	Users camped in cellA only	40
Gen_users_B	Users camped in cellB only	40
t_int_req	Inter request time	400 sec.
t_fail	Inter request time after fail	30 sec.
t_drop	Inter request time after drop	30 sec.
thr	Throughput	12.2 Kbit/sec.
OutageTime	Time to outage	36250 sec.

Table 1. Main parameters' values

and then an appropriate number of users (tokens) is moved from place `start_service` to place `Dropped`, representing the users whose service has been dropped due to an outage. Place `Dropped` will contain the same number of tokens as place `Discarded`. The “dropped” users will issue a new service request after a time defined by the exponential transition `T_drop`.

For the sake of readability, in Figure 5 we depict with a shaped circle the places shared with the architectural model, and with a rectangle the places shared with the users/arch model.

4.3 “usersA/arch1” model

In Figure 6 we depict the “usersA/arch1” model, that is the model representing the interactions between the class of generic users and the architecture of the first cell.

When the cell is down, place `Down` contains one token. Tokens in place `check_request` correspond to the service requests issued by the users that try to obtain an access slot. If the cell is down, the tokens are moved to place `Fail`, otherwise to place `ok`. The main components of the model are the input gates `Channel_allocation` and `Channel_deallocation`. The input gate `Channel_allocation` check if the admission control algorithm has been passed. If it has been passed ($\text{Mark}(\text{Channel}) > 0$) then a token is added to place `start_service`, otherwise there are no available traffic channels ($\text{Mark}(\text{Blocked}) > 0$) and then a token is added to place `Fail`. The input gate `Channel_deallocation` is in charge to free a traffic channel when a user has been served ($\text{Mark}(\text{end_service}) > 0$). In this case, a token is added to place `Users` and a token is removed from place `Channel_count`.

5 Results

Although the focus of this paper is not on the evaluation analysis, in this Section we sketch some results that we obtain through the solution of the models described in Section 4. A transient analysis has been performed, using the simulator provided by the Möbius tool [12]. Table 1 summarizes the settings for the main system parameters.

In Figure 7 we show the mean number of failed service requests at varying of time. This is clearly a QoS indicator since it corresponds to the mean number of unsatisfied users at varying of time. At steady-state (label `T0`), about 7 users in the network are unsatisfied, and it is about 7% of the total number of users camped in the overall network (7 out of 96). After the occurrence of the outage (label `T1`) this number rapidly increases until it reaches the new steady-state value of about 14 unsatisfied users,

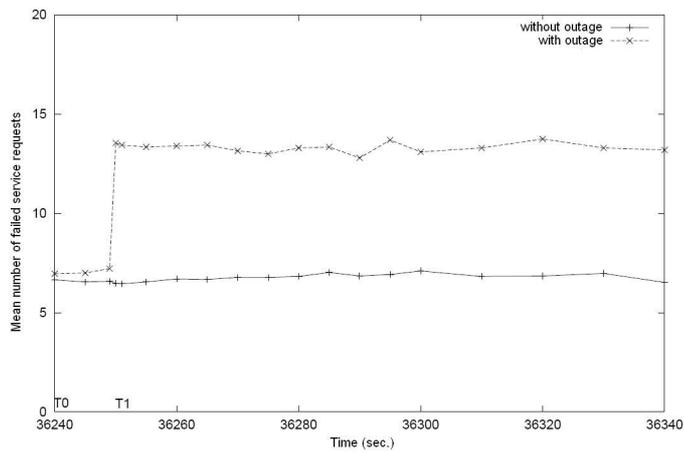


Figure 7. Mean number of failed service requests

thus the number of unsatisfied users is doubled.

6 Conclusions

In this paper we propose a general modular modeling approach for cellular systems. The approach is general in the sense that it is not domain-specific and then it is reusable for the analysis of several systems, including the mobile cellular networks. The feasibility of the proposed modeling approach is proved analyzing a case-study consisting of a UMTS network. The scenario is composed by two partially overlapping cells and we analyzed the QoS perceived by the users in the network in terms of mean number of failed service requests. Actually, thanks to the adopted modeling approach, we could easily modify the models in order to account for different network topologies, different types of outages (e.g. partial outages) and additional events like, for example, the repair of a failed component.

As future work, we are planning to use this modeling approach to analyze heterogeneous networks that combine, for example, both GPRS and UMTS cells, thus resembling very interesting scenarios as those treated in the context of the European Projects HIDENETS [13] and CRUTIAL [14].

7 Acknowledgements

The authors want to gratefully acknowledge the contribution given by Elisa Culicchi to the early phases of this work. This work has been partially supported by the European Community through the HIDENETS project.

Bibliography

- [1] ETSI, “Digital Cellular Telecommunications System (Phase 2+); General Packet Radio Service (GPRS); Service Description; Stage 2,” GSM 03.60 version 7.1.0, Release 1998.
- [2] ETSI TS 25.214 “Physical layer procedure (FDD)”, Release 1999, <http://www.3GPP.org>
- [3] ETSI TS 25.321 “MAC protocol specification”, Release 2002, <http://www.3GPP.org>
- [4] C. Betous-Almeida. *Construction et Affinement de modèles de sûreté de fonctionnement - application aux systèmes de contrôle-commande*, PhD Dissertation, Toulouse, 2002.
- [5] M. Rabah, & K. Kanoun. Performance Evaluation of Multipurpose Multiprocessor Systems: the “ Separation of Concerns” Approach. In *IEEE Transactions on Computers*, vol. 52, No. 2, 2003.
- [6] A. Bondavalli, M. Nelli, L. Simoncini, & G. Mongardi. Hierarchical modelling of complex control systems: dependability analysis of a railway interlocking. In *International Journal of Computer Systems Science & Engineering*, CRL Publishing Ltd, vol. 4, pages 249-261, 2001.
- [7] P. Lollini, A. Bondavalli, & F. Di Giandomenico. QoS analysis of a UMTS cell with different service classes. In *Proc. of the Fourth IASTED International Conference On Communication Systems and Networks (CSN)*, pages 55-60, Benidorm, Spain, September 12-14, 2005.
- [8] S. Porcarelli, F. Di Giandomenico, A. Bondavalli, M. Barbera, & I. Mura. Service-level availability estimation of GPRS. In *IEEE Transactions on Mobile Computing*, 2(3), pages 233-247, 2003.
- [9] ETSI TR 25.922 “Technical Specification Group Radio Access Network: Radio Resource Management Strategies”, Release 2005, <http://www.3GPP.org>
- [10] J. F. Meyer, A. Movaghar, & W. H. Sanders. Stochastic activity networks: structure, behavior, and application. In *Proc. International Workshop on Timed Petri Nets*, Torino, Italy, 1985, 106-115.
- [11] W. H. Sanders, & J. F. Meyer. Reduced base model construction methods for stochastic activity networks. In *IEEE Journal on Selected Areas in Communications*, 9(1), 1991, 25-36.
- [12] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, & P. G. Webster. The Mobius framework and its implementation. In *IEEE Transactions on Software Engineering*, 28(10), 2002, 956-969.
- [13] IST-2004-26979 HIDE NETS Project. “Highly DEpendable ip-based NETworks and Services”. <http://www.hidenets.aau.dk/>
- [14] IST-2004-27513 CRUTIAL Project. “CRITICAL UTility InfrastructurAL Resilience”. <http://crutial.cesiricerca.it/>

A System Dependability Modeling Framework using AADL and GSPNs

Ana-Elena Rugina, Karama Kanoun and Mohamed Kaâniche

LAAS-CNRS

Abstract

For efficiency and cost control reasons, system designers' will is to use an integrated set of methods and tools to describe specifications and designs, and also to perform dependability analyses. AADL (Architecture Analysis and Design Language) has proved to be efficient for architecture modeling. This paper presents a modeling framework allowing the generation of dependability-oriented analytical models from AADL models, to facilitate the evaluation of dependability measures, such as reliability or availability. We propose a stepwise approach for system dependability modeling using AADL. The AADL dependability model is transformed into a GSPN (Generalized Stochastic Petri Net) by applying model transformation rules. The resulting GSPN can be processed by existing tools. The modeling approach is illustrated on a small example.

1. Introduction

The increasing complexity of new-generation embedded systems raises major concerns in various critical application domains, in particular with respect to the validation and analysis of performance, timing and dependability-related requirements. Model-driven engineering approaches based on architecture description languages aimed at mastering this complexity at the design level have emerged and are more and more extensively used in industry. In particular, AADL (Architecture Analysis and Design Language) [1] has received an increasing interest during the last years. It has been recently developed and standardized under the auspices of the International Society of Automotive Engineers (SAE), to support the design and analysis of complex real-time safety-critical systems in avionics, automotive, space and other application domains. AADL provides a standardized textual and graphical notation, for describing software and hardware system architectures and functional interfaces, and for performing various types of analysis to determine the behavior and performance of the system being modeled. The language has been designed to be extensible to accommodate analyses that the core language does not support.

Besides describing the systems' behavior in the presence of faults, the developers are interested in obtaining quantitative measures of relevant dependability properties such as reliability, availability and safety. For pragmatic reasons, the system designers using an AADL-based engineering approach are interested in an integrated set of methods and tools to describe specifications and designs, and to perform dependability evaluations. The *AADL Error Model Annex* [2] has been recently defined. It complements the description capabilities of the core language by providing features with precise semantics to be used for describing dependability-related characteristics in AADL models (faults, failure modes and repair assumptions, error propagations, etc.). However, at the current stage, no methodology and guidelines are available to help the developers in the use of the proposed notations to describe complex dependability models reflecting real-life systems with multiple interactions and dependencies among components. One of the two objectives of this paper is to propose a structured method for AADL dependability model construction.

The AADL Error Model Annex mentions that stochastic automata such as fault trees and Markov chains can be generated from AADL specifications enriched with dependability-related information. Indeed, Markov chains are recognized to be powerful means for modeling system dependability taking into account dependencies between system components. Usually, Generalized Stochastic Petri Nets (GSPNs) are used to generate automatically Markov chains. In addition, GSPNs allow structural model verification and analysis, before the Markov chain generation. Such verification support facilities are very useful when dealing with large models. During the last decade, various approaches have been defined to support the systematic construction and validation of dependability models based on GSPNs and their extensions (see e.g. [3-5]). We propose to take advantage of such approaches in the context of an AADL-based engineering process, to i) build the dependability-oriented AADL model and to ii) generate dependability-oriented GSPN models from AADL models by model transformation. In this way, the complexity of GSPN model generation is hidden to users who are familiar with AADL and have a limited knowledge of GSPNs. The AADL and GSPN models are built iteratively, taking into account progressively the dependencies between the components, and validated at each iteration.

To summarize, the objectives of this paper are twofold: i) provide guidelines for a structured and stepwise approach for building AADL dependability models and ii) show examples of model transformation rules to generate GSPNs from AADL dependability models. The set of model transformation rules is meant to be the basis for the implementation of a model transformation tool completely transparent for the user. Such a tool can be interfaced with one of the existing GSPN processing tools (e.g., Surf-2 [6], Möbius [7], Sharpe [8], GreatSPN [9], SPNP [10]) to evaluate dependability/performability measures.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 presents the AADL concepts that are necessary for understanding our modeling approach. Section 4 gives an overview of our framework for system dependability modeling and evaluation using AADL and GSPNs. Section 5 presents examples of rules for transforming AADL into GSPN models. Section 6 applies our approach to a small example and Section 7 concludes the paper.

2. Background and related work

To the best of our knowledge there are no similar contributions in the current state of the art. Most of the published papers on analyses using AADL have focused on the extension of the language capabilities to support formal verifications. For example, the COTRE project [11] provides a design approach bridging the gap between formal verification techniques and requirements expressed in Architecture Description Languages. AADL system specifications can be imported in the newly defined COTRE language. A system specification in COTRE language can be transformed into timed automata, Time Petri nets or other analytical models. However, to the best of our knowledge, there are no similar contributions aiming at obtaining dependability-oriented quantitative evaluation models from COTRE specifications.

Considering the problem of generating dependability evaluation models from model-driven engineering approaches in a more general context, a significant amount of research has been carried out based on UML (Unified Modeling Language) [12]. For example, the European project HIDE ([13], [14]) proposed a method to automatically analyze and evaluate dependability attributes from UML models. It defined several model transformations: i) from structural and behavioral UML diagrams into GSPNs, Deterministic and Stochastic Petri Nets and Stochastic Reward Nets to evaluate dependability measures, ii) from UML statechart diagrams into Kripke structures for formal verification and iii) from UML sequence diagrams into Stochastic Reward Nets for performance analysis. Also, [15] proposes an algorithm to synthesize dynamic fault trees (DFT) from UML system models (a conjunction of class, object and deployment diagrams extended with

stereotypes and tagged values). Other interesting approaches have been developed, aiming at obtaining performance measures by transforming UML diagrams (activity diagrams in [16], sequence and statechart diagrams in [17]) into GSPNs.

AADL is different from UML, as in AADL the user deals with a single annotated architecture model. Thus, the modeling approaches mentioned above cannot be directly applied in our context of AADL. The modeling framework presented in this paper is complementary to the above initiatives and is aimed at ensuring a better integration of dependability evaluation techniques based on GSPNs into AADL-based engineering approaches.

3. AADL concepts

The AADL core language allows analyzing the impact of different architecture choices (such as scheduling policy or redundancy scheme) on a system's properties [18]. An architecture specification in AADL describes how components are combined in subsystems and how they interact. Architectures are described hierarchically. *Components* are the building blocks of AADL architectures. They are grouped into three categories: 1) software (process, subprogram, data, thread, thread group), 2) hardware (processor, memory, device, bus) and 3) composite (system). AADL components can be composed of subcomponents and interconnected through features (ports, subprogram calls, parameters) that specify how components interface each other. Each AADL component has two levels of description: the *component type* and the *component implementation*. The component type describes how the environment sees that component (i.e., its properties and features). One or more component implementations can be associated with the same component type, corresponding to different implementation structures of the component in terms of subcomponents, connections, subprogram calls and operational modes.

The AADL core language is designed to describe static architectures with operational modes for their components. However, it can be extended to associate additional information to the architecture. *AADL error models* are an extension intended to support (qualitative and quantitative) analyses of dependability attributes. The AADL Error Model Annex defines a sub-language to declare error models within an error annex library. The AADL architecture model serves as a skeleton for error model instances.

```

Error Model Type [basic]
error model basic
features
  Error_Free: initial error state;
  Failed: error state;
  Fail, Repair: error event;
end basic;

Error Model Implementation [basic.nominal]
error model implementation basic.nominal
transitions
  Error_Free-[Fail] -> Failed;
  Failed-[Repair] -> Error_Free;
properties
  Occurrence => Poisson  $\lambda$  applies to Fail;
  Occurrence => Poisson  $\mu$  applies to Repair;
end basic.nominal;

```

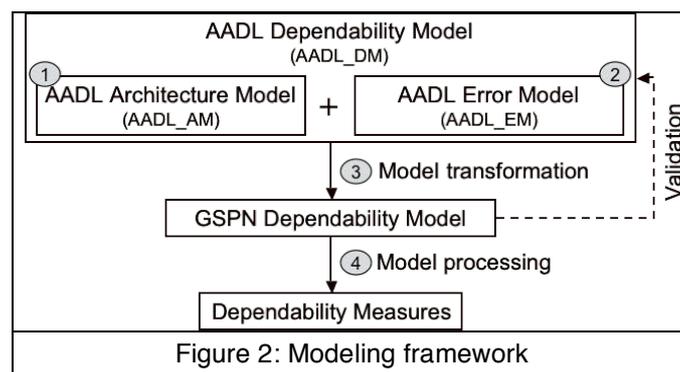
Figure 1: Basic error model

Component error models describe the behavior of the components to which they are associated, in the presence of internal faults and repair events, as well as in the presence of external propagations from the component’s environment. In the same way as for AADL components, error models have two levels of description: the *error model type* and the *error model implementation*. The error model type declares a set of error states, error events (internal to the component) and error propagations¹ (events that propagate, from one component to other components, through the connections and bindings between components of the architecture model). In addition, the user can declare Guard properties to control propagations.

Error model implementations declare transitions between states, triggered by events and propagations declared in the error model type and Occurrence properties that specify the arrival rate or the occurrence probability of events and propagations. Figure 1 shows a basic error model (without propagations).

Propagations have associated directions. The identifier *in* identifies incoming propagations while the identifier *out* identifies outgoing propagations. An *out* propagation occurs in a source error model according to a user-specified Occurrence property. The source error model sends the propagation out through all ports and bindings of the AADL component with which this error model is associated. Consequently an *out* propagation arrives to one or more error models associated with receiver components. If the receiver error models declare *in* propagations with the same name as the arriving *out* propagation, the *in* propagations can influence their behavior (i.e., they may trigger transitions between states and/or operational mode changes). All error models that receive the same *out* propagation and that declare name matching *in* propagations are influenced by this propagation simultaneously, i.e., state transitions and operational mode changes triggered by the *in* propagation (that matches the *out* propagation received) are simultaneous.

Guard properties (associated with ports, data components, and client and server subprograms) allow controlling propagations by means of Boolean expressions and predicates. For example, the *Guard_In* property defines Boolean expressions that specify how propagations arriving at a receiver component are translated or masked before impacting the stochastic automaton associated with it via the error model.



¹ In the AADL Error Model Annex specification, the terms “state”, “event” and “propagation” are associated with the term “**error**” in order to highlight that these features are specified in the AADL **Error** Model Annex. However, error states can also model error free states, error events can also model repair events and error propagations can model all kinds of notifications. In the rest of the paper we shall use the terms “state”, “event” and “propagation”. The associated term “error” will be omitted.

The system error model is defined as a composition of a set of concurrent finite stochastic automata corresponding to components. In the same way as the entire architecture, the system error model is described hierarchically. If both a container component (i.e., a component that contains subcomponents) and some of its subcomponents have error models, then the relationship between the error models must be declared. One can specify the state of a container component as i) a *function* of its subcomponents' states (i.e., the error model of the container component is *derived* from the error models of its subcomponents) or as ii) an *abstraction* of the behavior of its subcomponents in the presence of faults.

4. Overview of the modeling framework

For complex systems, the main difficulty for dependability model construction arises from dependencies between the system components. Dependencies can be of several types, identified in [4]: structural, functional or related to the fault-tolerance and to the maintenance strategies. As some components' behavior may depend on several others, a structured approach is needed to model dependencies in a systematic way, to avoid errors in the resulting model of the system and to facilitate its validation. In our approach, the AADL dependability-oriented model is built in a progressive and iterative way. More concretely, in a first iteration, we propose to build the model of the system's components, representing their behavior in the presence of their own faults and repair events only. The components are thus modeled as if they were *isolated* from their environment. In the following iterations, we introduce dependencies between the component models in an incremental manner.

An overview of our proposed iterative modeling framework, which can be decomposed in four main steps, is presented in Figure 2.

The **first step** is devoted to the modeling of the system architecture in AADL (in terms of components and operational modes of these components). The AADL architecture model (AADL_AM) may be available if it has been already built for other purposes.

The **second step** concerns the specification of the system behavior in the presence of faults through AADL error models (AADL_EMs) associated with components of the AADL_AM. The AADL_EM of the system is a composition of the set of AADL_EMs.

The AADL_AM and the AADL_EM of the system form a dependability-oriented AADL model, referred to as the *AADL dependability model (AADL_DM)* in the rest of the paper.

The **third step** aims at building a GSPN dependability model, from the AADL dependability model, based on model transformation rules.

The **fourth step** is devoted to the GSPN model processing that aims at evaluating quantitative measures characterizing dependability attributes. This step is entirely based on existing processing algorithms and tools. Therefore, it is not considered in this paper.

To obtain the AADL_DM, the user must perform the first and second steps described above. The third step is intended to be automatic in order to hide the complexity of the GSPN to the user.

The iterative approach can be applied to the second step only or to the second and third steps together. In the latter case, semantic validation based on the GSPN model, after each iteration, is helpful to identify specification errors in the AADL_DM.

In the rest of the section we first give guidelines for the AADL_DM construction and then we provide an overview of the AADL to GSPN transformation.

4.1. The AADL dependability model construction

The AADL_EM of the system is built in several iterations. In the first iteration, we build the basic AADL_EMs associated with components, modeling their behavior in the presence of their own faults and repair events only. In the following iterations, dependencies between basic AADL_EMs are introduced progressively. Purely structural and functional dependencies must be included before the fault tolerance and maintenance dependencies. Fault tolerance and maintenance dependencies may have an impact on the system's structure. Lastly, the user can define derived AADL_EMs for container components. In this way, the final model represents the behavior of each component not only in the presence of its own faults and repair events, but also in its environment, i.e., faults and repair events in components with which it interacts.

It is noteworthy that not all the details of the AADL_AM are necessary for the AADL_DM. Only components that have associated AADL_EMs and all connections and bindings between them are necessary.

To illustrate the proposed approach, the rest of this subsection presents successively guidelines for modeling i) an architecture-based dependency (structural, functional or fault tolerance), ii) a maintenance dependency and iii) a hierarchical system depending on its subcomponents.

Architecture-based dependency

The dependency is modeled in the AADL_EMs associated with dependent components, by specifying respectively outgoing and incoming propagations and their impact on the corresponding AADL_EM. An example is shown in Figure 3: *Component 1* sends data to *Component 2*, thus we assume that, at the AADL_EM level, the behavior of *Component 2* depends on that of *Component 1*. Let us assume that the AADL_EM of Figure 1 can be associated both to *Component 1* and to *Component 2* to model the behavior of each of these two components as if they were isolated. To model a dependency between them, we need to add:

- In the AADL_EM associated with *Component 1*: i) an out propagation declaration in the type and ii) its associated Occurrence property and an AADL transition triggered by it in the implementation.
- In the AADL_EM associated with *Component 2*: i) the declaration of the corresponding in propagation in the type and ii) an AADL transition triggered by it in the implementation.

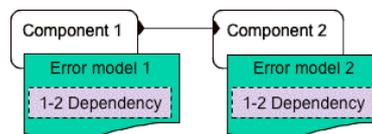


Figure 3: Architecture-based dependency

Maintenance dependency

Maintenance dependencies need to be described when repair facilities are shared between components or when the maintenance or repair activity of some components has to be carried out according to a given order or a specified strategy (i.e., software can be restarted only if the hardware is available).

Components that are not dependent at architectural level may become dependent due to the maintenance strategy. Thus, the AADL_AM might need some adjustments to support the description of dependencies related to the maintenance strategy. As AADL_EMs interact only via propagations through architectural features (i.e., connections, bindings), the maintenance dependency between components' AADL_EMs must also be supported by the AADL_AM. This means that besides the system architecture components, we may need to add an AADL_AM component allowing to describe the maintenance strategy. Figure 4-a shows an example of AADL_DM. In this architecture, *Component 3* and *Component 4* do not interact at the AADL architecture level, as there is no architecture-based dependency between them. However, if we assume that they share one repairman, the maintenance strategy has to be taken into account in the AADL_EM of the system. Thus, it is necessary to represent the repairman at the AADL_AM level, as shown in Figure 4-b in order to model explicitly the maintenance dependency between *Component 3* and *Component 4*.

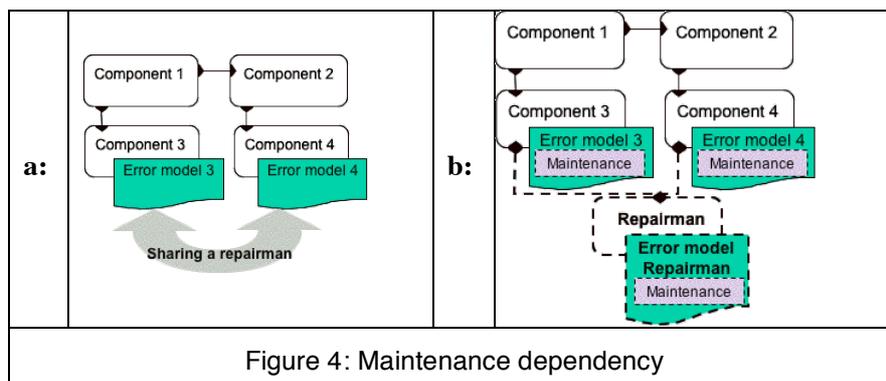


Figure 4: Maintenance dependency

Hierarchical systems

Figure 5 shows an example of a hierarchical system where *Component 1* is a container component and the AADL_EM associated with it is *derived* from the AADL_EMs of its subcomponents. In this case, the states of *Component 1* are defined by a user-specified function of the states of its subcomponents.

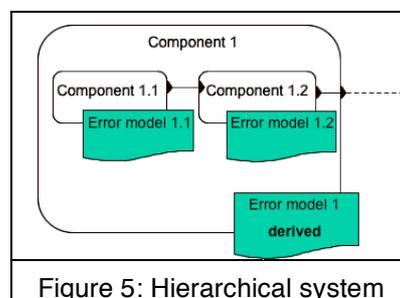


Figure 5: Hierarchical system

The AADL Error Model Annex offers the possibility of declaring abstract AADL_EMs for container components. This option can be useful to abstract away modeling details in case an AADL_AM with too detailed AADL_EMs associated with basic components does exist for other purposes. Issues linked to the relationship between *abstract* and *concrete* stochastic automata models obtained from AADL_EMs have

been mentioned in [19]. If abstract AADL_EMs are declared, the corresponding container components are seen as black boxes (i.e., the detailed subcomponents' AADL_EMs are not part of the AADL_DM).

4.2. AADL to GSPN model transformation

The GSPN obtained by model transformation is formed of several GSPN blocks connected through arcs. A block is a sub net describing either the component's behavior in the presence of its own faults and repair events (*component net*), or a dependency (*dependency net*). In the AADL_DM, each dependency is modeled as part of each of the AADL_EMs involved in the dependency. GSPN dependency nets are obtained from information concerning a particular dependency existing in (at least) two dependent AADL_EMs. The global GSPN contains one block for the behavior of each component in the presence of its own faults and repair events, and one block for each dependency between components. A component having a derived AADL_EM is transformed as follows:

- Each of its subcomponents having an AADL_EM is transformed into a component net,
- The derived state mapping expression is transformed into a *derived dependency net*, as the states of the component depend only on the states of its subcomponents.

Figure 6 shows the modular GSPN obtained by model transformation from the AADL_DM shown in Figure 3. The dependency between *Component 1* and *Component 2* is represented in this GSPN model by a separate block. The arrows that link the GSPN blocks represent the direction of the dependency. Here, the direction is the same as the direction of the connections in the AADL_AM. In a more general case, when architectural connections are bi-directional, the dependency at AADL_EM level may be uni or bi-directional depending on the explicit direction of propagations.

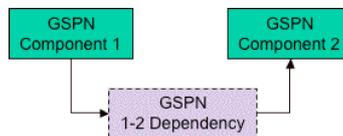


Figure 6: GSPN block representation

The modular structure of the GSPN allows the user to validate the model progressively, as the GSPN is enriched with a block each time the second step is iterated, i.e., a new dependency is added in the AADL_EM of the system. So, if validation problems arise at GSPN level during iteration *i*, only the part of the current AADL_EM corresponding to iteration *i* is questioned.

5. Transformation rules

In the next three subsections we present successively AADL to GSPN transformation rules for i) isolated components, ii) a set of dependent components and iii) hierarchical systems. All transformation rules are defined to ensure by construction the various properties for the resulting GSPN to be syntactically and semantically correct. They are aimed to be systematic in order to prepare the transformation automation. Also, the resulting GSPN is generic and tool-independent. It is worth noting that this section only presents a few examples of the transformation rules. A more complete set of rules is presented in [20].

5.1. Isolated components

In the case of an isolated component or in the case of a set of independent components, the AADL to GSPN transformation is rather straightforward, as an AADL_EM represents a stochastic automaton, as shown in the example from Figure 1. Table 1 shows the basic transformation rules.

The corresponding component block is formed of places and transitions. The number of tokens in a component block is always one, as a component can only be in one state.

Table 1: Basic AADL error model to GSPN transformation rules

AADL error model element	GSPN element		
State	Place		
Initial state	Token in the corresponding place		
Event	GSPN transition (timed or immediate)		
Occurrence property of an event	Distribution or probability characterizing the occurrence of associated GSPN transition		 Timed
			 Immediate
AADL transition (Source_State-[Event] -> Destination_State)	Arcs connecting places (corresponding to AADL Source_State and Destination_State) via GSPN transition (corresponding to AADL Event)		

5.2. Set of dependent components

As described in Section 0, dependencies between components are expressed in AADL through i) name matching propagations and ii) Guard properties, which are propagation control mechanisms. There are four Guard properties: *Guard_In*, *Guard_Out*, *Guard_Event* and *Guard_Transition*. *Guard_In* and *Guard_Out* are used to filter respectively incoming and outgoing propagations while *Guard_Event* and *Guard_Transition* are used to link the AADL_EM specification to the architecture model operational modes (i.e., to specify mode changes triggered by the propagations). For didactical reasons, in this section, we only focus on the AADL to GSPN transformation rules for name matching propagations.

We first present an example of a pair of AADL_EMs that declare name matching propagations in Figure 7. Then we illustrate two possible transformation rules on this example. Finally, we generalize and we discuss the advantages of each of the two rules.

We assume that the two AADL_EMs of Figure 7 are associated with AADL components that are connected or bound one to the other.

- The AADL_EM at the left corresponds to the propagation sender. It declares two states: *Error_Free* (initial state) and *Failed*. The occurrence of the event *Fail* triggers the AADL transition between the source state *Error_Free* and the destination state *Failed*. Also, this AADL_EM declares one out propagation, named *Sender_Failed*. This notification is propagated out from the state *Failed* (with a probability p) and does not affect the state of the sender component.
- The AADL_EM at the right corresponds to the propagation receiver. It is identical to the sender AADL_EM, except for the direction of the propagation *Sender_Failed*, which is an in propagation in

the receiver AADL_EM. If the receiver AADL_EM receives the propagation *Sender_Failed* when being in the state *Error_Free*, it moves to the state *Failed*.

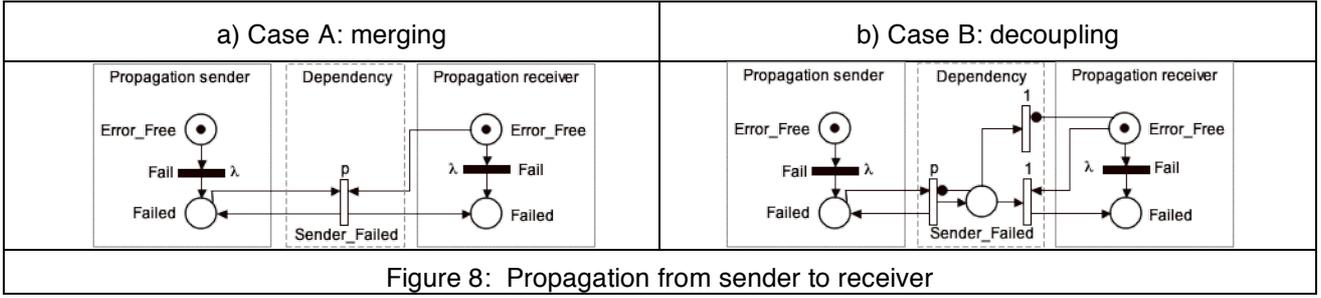
<p style="text-align: center;">Error Model Type [Sender_example]</p> <pre>error model Sender_example features Error_Free: initial error state; Failed: error state; Fail: error event; Sender_Failed: out error propagation; end Sender_example;</pre>	<p style="text-align: center;">Error Model Type [Receiver_example]</p> <pre>error model Receiver_example features Error_Free: initial error state; Failed: error state; Fail: error event; Sender_Failed: in error propagation; end Receiver_example;</pre>
<p style="text-align: center;">Error Model Implementation [Sender_example.basic]</p> <pre>error model implementation Sender_example.basic transitions Error_Free- [Fail] -> Failed; Failed- [out Sender_Failed] -> Failed; properties Occurrence=>poisson λ applies to Fail; Occurrence => fixed p applies to Sender_Failed; end Sender_example.basic;</pre>	<p style="text-align: center;">Error Model Implementation [Receiver_example.basic]</p> <pre>error model implementation Receiver_example.basic transitions Error_Free- [Fail] -> Failed; Error_Free- [in Sender_Failed] -> Failed; properties Occurrence=>poisson λ applies to Fail; end Receiver_example.basic;</pre>

Figure 7: Sender and Receiver – name-matching propagations

When the out propagation *Sender_Failed* occurs in the sender AADL_EM, it propagates out of the component to which this AADL_EM is associated and reaches the receiver AADL_EM. The AADL transitions triggered respectively by the out propagation in the sender AADL_EM and by the in propagation in the receiver AADL_EM are simultaneous. The propagation *Sender_Failed* does not induce a state change in the sender AADL_EM. It is only a consequence of the event *Fail* on the environment.

Two possible AADL to GSPN transformation rules are presented hereafter. The selection of the more appropriate one will depend on the number of AADL transitions triggered by the (in and out) propagation.

- Case A: the transformation rule consists in **merging** the out propagation from the sender AADL_EM to the in propagation from the receiver AADL_EM in one GSPN transition, as shown in Figure 8-a. The Occurrence property of the out propagation characterizes the occurrence of the GSPN transition.
- Case B: the transformation rule consists in **decoupling** the in and out propagations in the GSPN through an intermediary place, as shown in Figure 8-b. A token arrives in the newly introduced place when a GSPN transition corresponding to the out propagation occurs. This token is evacuated through an immediate GSPN transition of probability 1 that has either i) an input arc from a place corresponding to a source state for an AADL transition triggered by the in propagation in the receiver AADL_EM or ii) input inhibitor arcs coming from all places corresponding to such source states. Thus, the token from the intermediary place is always evacuated and the consequences of the out propagation are not inhibited if the receiver is not in a source state for transitions triggered by the corresponding in propagation.



In the most general case, an *out* propagation declared in a propagation sender AADL_EM could trigger n AADL transitions in this same AADL_EM (i.e. a particular propagation could be propagated out from multiple states). Matching *in* propagations could be declared in $r \geq 2$ propagation receiver AADL_EMs and trigger m_j AADL transitions in each j ($j = 1 \dots r$) receiver AADL_EM.

The number of GSPN transitions N_{tr} needed to describe the AADL propagation is given by:

$$\text{Case A: } N_{tr} = n * \left(\prod_{j=1}^r (m_j + 1) - 1 \right), \forall r \geq 1 \quad \text{Case B: } N_{tr} = n + \prod_{j=1}^r (m_j + 1), \forall r \geq 1$$

Case B is best suited for $n=2$ and $m_j > 3$ ($r = 1$) and for $n > 2$ and at least one $m_j > 2$ (for $r \geq 2$).

5.3. Hierarchical systems

The behavior of a system in the presence of faults can be described, using derived AADL_EMs, in terms of *global states* depending on the states of its subcomponents. Derived AADL_EMs use Boolean expressions similar to those used in the *Guard* properties. These expressions determine the state of the derived AADL_EM (i.e., the global states).

Global states of the system correspond to places in the GSPN. The Boolean expression is transformed into a set of immediate GSPN transitions of probability 1. These transitions are connected through arcs to places that correspond to states of the subcomponents and to global states of the system. Only one place corresponding to a global state can be marked at a given time. Thus, the number of GSPN transitions corresponding to an atomic Boolean expression is equal to $n_g - 1$ (n_g being the number of places corresponding to global states). Each GSPN transition has an input arc coming from a place corresponding to a global state (which is emptied when the transition is fired). Initially, a token is placed in the place that corresponds to the global initial state of the system. This global initial state is determined from the initial states of the system's subcomponents.

An example is given in Figure 9, which shows the implementation of an AADL system component *A* with two subcomponents named *A1* and *A2*.

The *Derived_State_Mapping* expression specifies that the system is *Error_Free* if both its subcomponents are *Error_Free*, and *Failed* otherwise. Figure 10 shows the GSPN obtained after transformation of this derived AADL_EM. The place *Error_Free* corresponding to component *A* is marked if the places corresponding to *Error_Free* states for *A1* and *A2* are marked. Otherwise (i.e., at least one of the two places that correspond to *Error_Free* states for *A1* and *A2* is not marked), the place *Failed* of the derived AADL_EM is marked. Note that, *A* cannot be simultaneously in states *Error_Free* and *Failed*. One GSPN transition corresponds to each atomic Boolean expression (the expression *when others* is formed of three atomic Boolean expressions).

```

system implementation A.nominal
subcomponents
  A1: system hardware.nominal;
  A2: system software.nominal;
annex Error_Model {**
  Model => forA.basic;
  Derived_State_Mapping =>
    Error_Free when
      (A1[Error_Free] and A2[Error_Free]),
    Failed when others;
**};
end A.nominal;

```

Figure 9: Example of derived error model

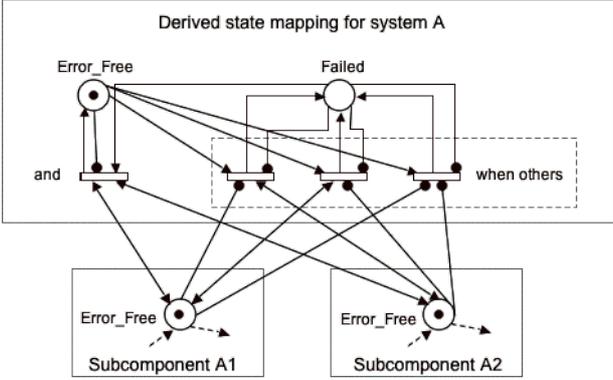


Figure 10: GSPN modeling of the derived state mapping

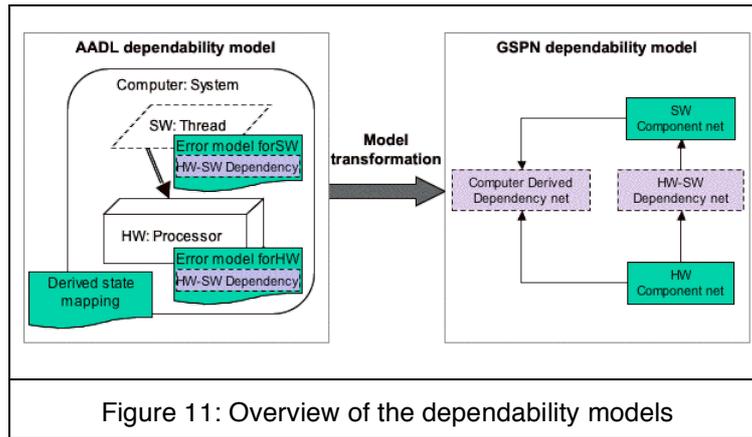
It is noteworthy that the GSPN obtained above is tool-independent. However, some evolved GSPN processing tools are able to process Boolean expressions associated with GSPN transitions. In this case, the transformation rule can be simplified, as Boolean expressions of the `Derived_State_Mapping` expression can be directly placed on GSPN transitions.

6. Didactical example

In this section we illustrate our modeling approach. We use the transformation rules described in section 0. Due to space limitations we only present a toy example with one dependency. A more realistic one is presented in [20].

The system considered here represents a simple computer. The system is formed of two subcomponents: a hardware subcomponent (processor) and a software subcomponent (thread) bound to the hardware. Permanent faults in the hardware cause hardware failures and require hardware maintenance. Temporary faults in the hardware do not require hardware maintenance but they can cause error propagations to the software on top of it. This leads to a structural dependency between the hardware and the software, as a processor can propagate errors to threads bound to it. The overall system is considered to be working if both the hardware and the software are error free, and failed otherwise. The overview of the AADL and GSPN dependability models is given in Figure 11 using the AADL graphical notation for the thread, the processor and the binding of the thread to the processor.

The following two subsections present respectively the AADL_EM construction and the AADL to GSPN model transformation.



6.1. AADL_EM of the system

Figure 12 shows at the left the AADL_EM associated with the hardware component and at the right the AADL_EM associated with the software component.

The AADL_EM type *forHW* declares i) three states: *HW_Err_Free* (initial state), *HW_Err* and *HW_Failed*, ii) four events: *Temp_Fault*, *Perm_Fault*, *Disappear*, *Repair* and iii) one out propagation: *HW_Temp*. The AADL_EM implementation *forHW.basic* declares AADL transitions between states declared in the AADL_EM type *forHW*. If a temporary fault occurs (*Temp_Fault*), the hardware moves from the state *HW_Err_Free* to the state *HW_Err*. A hardware error disappears (*Disappear*) after a while and the hardware returns to state *HW_Err_Free*. If a permanent fault occurs (*Perm_Fault*), the hardware moves from state *HW_Err_Free* to state *HW_Failed*. A failure requires repairing (*Repair*) the hardware. Occurrence properties are associated with all events and out propagations. An error caused by a temporary fault propagates out (*HW_Temp*) with a probability $p1$.

The AADL_EM type *forSW* declares i) three states: *SW_Err_Free* (initial state), *SW_Err* and *SW_Failed*, ii) four events: *Fault*, *Recover*, *Non_Recover*, *Restart* and iii) one in propagation: *HW_Temp* (name matching to the out propagation from the *forHW* AADL_EM type). The AADL_EM implementation *forSW.basic* declares AADL transitions between states declared in the AADL_EM type *forSW*. When a *Fault* occurs, the software moves from state *SW_Err_Free* to state *SW_Err*. In some cases the error can be recovered (*Recover*) and the software returns to state *SW_Err_Free*. Otherwise (*Non_Recover*) the software moves to state *SW_Failed* and it needs to be restarted (*Restart*). Occurrence properties are associated with all events. An incoming propagation caused by a temporary fault in the hardware (*HW_Temp*) causes an AADL transition from state *SW_Err_Free* to state *SW_Err*. The error is then dealt with as if it were caused by a software fault.

<pre> Error Model Type [forHW] error model forHW features -- iteration 1 HW_Err_Free: initial error state; HW_Err, HW_Failed: error state; Temp_Fault, Perm_Fault, Disappear, Repair: error event; -- iteration 2 (HW-SW dependency) HW_Temp: out error propagation; end forHW; </pre>	<pre> Error Model Type [forSW] error model forSW features -- iteration 1 SW_Err_Free: initial error state; SW_Err, SW_Failed: error state; Fault, Recover, Non_Recover, Restart: error event; -- iteration 2 (HW-SW dependency) HW_Temp: in error propagation; end forSW; </pre>
<pre> Error Model Implementation [forHW.basic] error model implementation forHW.basic transitions -- iteration 1 HW_Err_Free-[Temp_Fault] -> HW_Err; HW_Err_Free-[Perm_Fault] -> HW_Failed; HW_Err-[Disappear] -> HW_Err_Free; HW_Failed-[Repair] -> HW_Err_Free; -- iteration 2 (HW-SW dependency) HW_Err-[out HW_Temp] -> HW_Failed; properties -- iteration 1 Occurrence => poisson λ1 applies to Temp_Fault; Occurrence => poisson λ2 applies to Perm_Fault; Occurrence => poisson μ1 applies to Disappear; Occurrence => poisson μ2 applies to Repair; -- iteration 2 (HW-SW dependency) Occurrence => fixed p1 applies to HW_Temp; end forHW.basic; </pre>	<pre> Error Model Implementation [forSW.basic] error model implementation forSW.basic transitions -- iteration 1 SW_Err_Free-[Fault] -> SW_Err; SW_Err-[Recover] -> SW_Err_Free; SW_Err-[Non_Recover] -> SW_Failed; SW_Failed-[Restart] -> SW_Err_Free; -- iteration 2 (HW-SW dependency) SW_Err_Free-[in HW_Temp] -> SW_Err; properties -- iteration 1 Occurrence => poisson λ3 applies to Fault; Occurrence => fixed p2 applies to Recover; Occurrence => fixed p3 applies to Non_Recover; Occurrence => poisson μ3 applies to Restart; end forSW.basic; </pre>

Figure 12: Error models *forHW* and *forSW*

The derived AADL_EM for the system component specifies that the system is error free when both the hardware and the software are error free, and failed otherwise. This derived state mapping has already been described in Figure 9.

The AADL_EM construction is accomplished in three iterations: states and events (with associated Occurrence properties) are declared together with transitions triggered by these events in a first iteration. The *HW_Temp* propagation together with its stochastic property and triggered transitions is introduced in a second iteration to explicit the dependency between the hardware and the software, as highlighted in Figure 12. The derived AADL_EM for the system component is added in a third iteration.

6.2. Model transformation

As the previous step, the model transformation is accomplished in three iterations. The resulting GSPN is shown in Figure 13. States and AADL transitions triggered by events are transformed into places and GSPN transitions belonging to HW and SW component nets. AADL transitions triggered by propagations are

transformed into GSPN transitions that form the HW-SW dependency net. The derived AADL_EM for the system component is transformed into the Computer derived dependency net.

For clarity reasons, we modeled here only the structural dependency between the software and the hardware. We did not show the maintenance dependency between these two components (i.e., the fact that the software cannot be restarted if the hardware is failed).

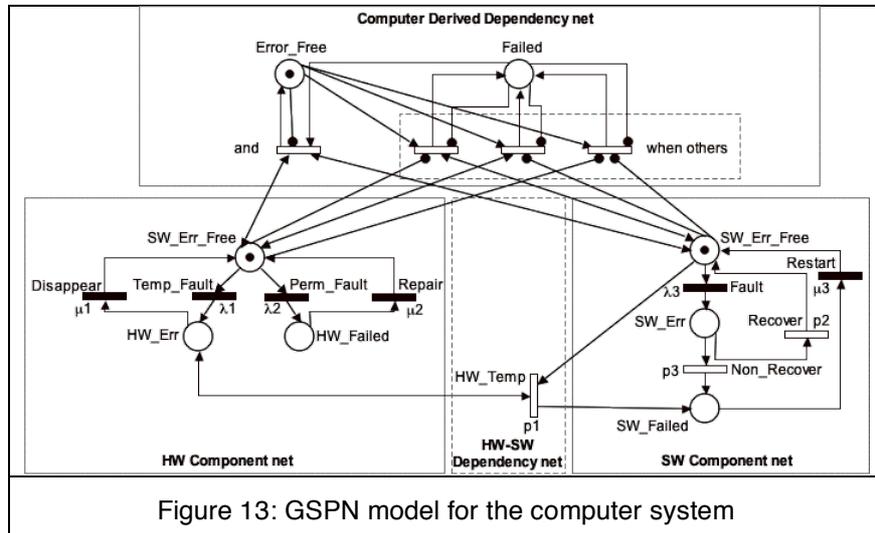


Figure 13: GSPN model for the computer system

7. Conclusion

This paper presented a stepwise approach for system dependability modeling using AADL and GSPNs. The aim of this approach is to hide the complexity of traditional analytical models to end-users acquainted with AADL. In this way, we ease the task of evaluating dependability measures. Our approach assists the user in the structured construction of the AADL_DM (i.e., architecture model + dependability-related information) that is transformed into a GSPN to be processed by existing tools. To support and trace model evolution, this approach proposes that the user builds the AADL_DM iteratively. Components' behaviors in the presence of faults are modeled in the first iteration as if they were isolated. Then, each iteration introduces a new dependency between system's components in the AADL_DM. The AADL to GSPN model transformation is meant to be transparent to the user. Thus, it is based on rigorous and systematic rules aimed at supporting tool-based transformation automation. The model transformation can be performed iteratively, each time the AADL_DM is enriched. In this way, the GSPN model can be validated progressively (hence the corresponding AADL architecture and error models can be validated progressively and corrected accordingly, if required). Finally, we illustrated the proposed approach on a toy example with one dependency. However, we have applied this approach to a complex-enough system, to assess its feasibility in [20]. In this paper, we have shown the principles of the transformation and some of the rules. The work in progress concerns the completion of the set of rules. Future work will focus on implementing a model transformation tool that can be easily integrated into AADL and GSPN based tools.

Acknowledgement. This work was partially supported by 1) the European Commission (European integrated project ASSERT No. IST 004033 and network of excellence ReSIST No. IST 026764), and 2) the European Social Fund.

References

- [1] SAE-AS5506, "Architecture Analysis and Design Language," Society of Automotive Engineers, Warrendale, PA 2004.
- [2] AADL-WG, "AADL Error Model Annex," submitted for formal ballot in September 2005. Copies can be asked by e-mail to info@aadl.info.
- [3] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and I. Mura, "Dependability Modeling and Evaluation of multiple-phased systems, using DEEM," *IEEE Transactions on Reliability*, vol. 53, pp. 509-522, 2004.
- [4] K. Kanoun and M. Borrel, "Fault-tolerant systems dependability. Explicit modeling of hardware and software component-interactions," *IEEE Transactions on Reliability*, vol. 49, pp. 363-376, 2000.
- [5] S. Bernadi, A. Bobbio, and S. Donatelli, "Petri Nets and Dependability," in *Lectures on Concurrency and Petri Nets*, vol. 3098, W. Reisig and G. Rozenberg ed: Springer-Verlag - LNCS, 2004, pp. 125-179.
- [6] C. Béounes, et al., "Surf-2: a program for dependability evaluation of complex hardware and software systems", 23rd IEEE International Symposium on Fault Tolerant Computing, Toulouse, France, pp. 668-673, 1993.
- [7] D. D. Deavours, et al., "The Mobius Framework and its Implementation," *IEEE Transactions on Software Engineering*, vol. 28, pp. 956-969, 2002.
- [8] C. Hirel, R. Sahner, X. Zang, and K. Trivedi, "Reliability and performability modeling using SHARPE 2000", 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools, Schaumburg, IL, USA, 1789, pp. 345-349, 2000.
- [9] S. Bernadi, et al., "GreatSPN in the new millenium", Tool Session of 9th Int. Workshop on Petri Nets and Performance Models, Aachen, Germany, 2001.
- [10] G. Ciardo and K. S. Trivedi, "SPNP: The Stochastic Petri Net Package (Version 3.1)", 1st Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'93), San Diego, CA, USA, pp. 390-391, 1993.
- [11] J.-M. Farines, et al., "The Cotre project: rigorous software development for real time systems in avionics", 27th IFAC/IFIP/IEEE Workshop on Real Time Programming Zielona Gora, Poland, 2003.
- [12] OMG, "Unified Modelling Language Specification: version 2.0," <http://www.omg.org> October 2004.
- [13] I. Majzik and A. Bondavalli, "Automatic Dependability Modeling of Systems Described in UML", International Symposium on Software Reliability Engineering (ISSRE), 1998.
- [14] A. Bondavalli, et al., "Dependability Analysis in the Early Phases of UML Based System Design," *International Journal of Computer Systems - Science & Engineering*, vol. 16, pp. 265-275, 2001
- [15] G. J. Pai and J. Bechta Dugan, "Automatic Synthesis of Dynamic Fault Trees from UML System Models", 13th International Symposium on Software Reliability Engineering (ISSRE'02), Annapolis, USA, pp. 243-254, 2002.
- [16] J. P. López-Grao, J. Merseguer, and J. Campos, "From UML Activity Diagrams To Stochastic Petri Nets: Application TO Software Performance Engineering", 4th International Workshop on Software and Performance (WOSP'04), Redwood City, California, USA, pp. 25-36, 2004.
- [17] S. Bernardi, S. Donatelli, and J. Merseguer, "From UML Sequence Diagrams and Statecharts to analysable Petri Net models", 3rd Int. Workshop on Software and Performance (WOSP 2002), Rome, Italy, pp. 35-45, 2002.
- [18] P. H. Feiler, D. P. Gluch, J. J. Hudak, and B. A. Lewis, "Pattern-Based Analysis of an Embedded Real-time System Architecture", 18th IFIP World Computer Congress, ADL Workshop, Toulouse, France, ADL Workshop, pp. 83-91, 2004.
- [19] P. Binns and S. Vestal, "Hierarchical composition and abstraction in architecture models", 18th IFIP World Computer Congress, ADL Workshop, Toulouse, France, ADL Workshop, pp. 43-52, 2004.
- [20] A. E. Rugina, K. Kanoun, and M. Kaâniche, "AADL-based Dependability Modelling," LAAS-CNRS Research Report n°06209, April 2006, www.laas.fr/~aerugina/AADLbasedDepModel.pdf.

Diversity for fault tolerance: effects of “dependence” and common factors in software development

Kizito Salako*, Lorenzo Strigini[†]
 Centre for Software Reliability
 City University
 Northampton Square, London EC1V-0HB
 Email: *kizito@csr.city.ac.uk, [†]strigini@csr.city.ac.uk

Abstract

Fault tolerance via diverse redundancy, with multiple “versions” of a system in a redundant configuration, is a naturally attractive defence against design faults. Its effectiveness is measured by how unlikely the common failures are that cause the whole redundant system to fail. It is well known that diversity cannot guarantee statistical independence between failures of the diverse redundant versions; questions of practical interest for development decisions concern the levels of system reliability to be expected, depending on whether diversity is used and how diverse developments are managed. The effect of diverse development of versions can be modelled, at a rather abstract level, and the models offer useful insight on how to seek effective diversification, recognising fallacies that may arise when addressing these subtle issues by “expert judgement” alone. However, the models published so far rely on a strong assumption of *independent sampling* of the possible versions, roughly representing a process in which the development processes of the diverse versions are kept rigorously isolated from each other: an “ideal” process according to many opinions. This “ideal” situation is unlikely in practice. We discuss under which circumstances the assumption is acceptable and the effects of relaxing it. We describe a generalised model allowing for many sources of dependence between the developments. Our discussion clarifies various aspects of how the management of development processes affects the effectiveness of diversity. e.g. the effects of different verification and debugging regimes. These have been discussed in the literature, but we provide here a way of stating the arguments in more rigorous terms, resolving some specific questions that may arise and giving some clarity regarding others.

I. INTRODUCTION

A defence against design faults in all kinds of systems is redundancy with diversity. In its simplest form, this means that a system is built out of a set of subsystems (known as *versions*, *channels*, *lanes*), which perform the same or equivalent functions and are connected in a “parallel redundant” (1-out-of-N) or a voted scheme¹. The rationale for such designs, instead of similar fault-tolerant designs that use multiple copies of the same subsystem, is that these multiple copies would contain the same design faults: any circumstances in which one of them were to fail would tend to cause the other copies to fail as well, possibly with results that, despite being incorrect, are plausible and consistent and thus cannot be recognised as failures. Diversity eliminates the certainty of design faults being reproduced identically in all channels of the redundant system; one can hope that any faults (rare, given good quality development) will be unlikely to be similar between channels, causing them not to fail identically in exactly the same situations. This low probability of common faults can be sought by seeking “independence” between the developments of the multiple versions. For instance,

- for custom-developed components, development teams work separately, making their separate design choices (and possible mistakes), within the constraints of the specifications and general project management directives. These directives may also be specifically geared at “forcing” more diversity, e.g. mandating different architectures or different development methods [1], [2], [3].
- when re-using pre-existing components for the diverse channels, one can seek assurance that the developments were indeed separate and, for instance, did not rely on common component libraries or designs.

We will refer to a scenario of diversity between *software* versions, since most previous literature in computing refers to this scenario; we will show later on how our method can be applied more generally.

The difficult question is how effective diversity is, as a function of how it is obtained, so that one can decide when to use diverse designs and how to manage their development. As usual in software engineering, experimental results are hard to generalise, especially to high-reliability systems, and these questions have generated lively debates, in which positions have

This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible.

¹A parallel redundant (1-out-of-N) system is one in which correct system functioning is assured provided at least one channel functions correctly; in a voted system, correct system functioning is assured if a majority of channels function correctly. Many other architectures are possible, but here we are interested in the simplest practical scenario where evaluation problems arise.

been mostly supported by appealing to experience and individual judgement. We attempt a rigorous probabilistic description of the issues involved, seeking the usual benefits of clarifying the assumptions used separating questions that require an empirical answer from those that can be answered by deduction, and providing useful insight.

There is little one can say a priori about the probability of common failure for a *specific* pair of versions. Some pairs may have no faults that lead to failures on the same demand; in some other pairs, every time one version fails the other one will fail as well. But can we at least predict something about the *average* results of applying diversity in a certain system?

One of the early questions was thus: will the average pair of versions behave like a pair of two average versions failing independently? ² The famous experiment by Knight and Leveson [4] refuted this conjecture: this “independence on average” property did not apply to the specific population of versions that their subject programmers developed, hence cannot be assumed to hold in general. On average, a pair of versions failed together with far higher probability than the square of the average *pdf* of individual versions (though far less frequently than the average individual version). This leads to the conjecture that the general law in diverse systems may be, unfortunately, one of positive correlation - on average - between version failures. Experimental evidence was not enough to support or refute such general claims. Probabilistic modelling offered a way of understanding what may be going on in diverse development. The breakthrough was due to Eckhardt and Lee [5]. The bases of their approach were:

- from the viewpoint of reliability, a program can be completely described by its behaviour - success or failure - on every possible demand;
- since the process of developing a program is itself subject to variation, so that one cannot know in advance (or even, in practice, after delivery) exactly which program will result from it, from the crucial viewpoint of which faults it contains, this development process can be modelled as a process of *random sampling*, which selects one program from the population of all possible programs. The visible properties of the development (system specifications, methods used, choice of developers) do not determine *exactly which program* is created, but they determine the *probabilities* of it being any specific one;
- some demands are more difficult for the developers to treat correctly than others. One can formally model this “difficulty” of each particular demand via the probability of a program, “randomly” chosen by the development process, failing on that specific demand.

In this modelling framework, the reputedly ideal conditions of complete isolation between the developments of the various versions are represented by the assumption that each program version is selected (sampled) independently of the selection of any other. Eckhardt and Lee [5] then showed that, if all versions are produced *independently* by identical development processes, “positive correlation on average” is inevitable, unless (implausibly) all demands have identical difficulty. Later, Littlewood and Miller [6] pointed out that each version may be developed by a *different* process: this is indeed the purpose of “forcing” diversity. With this less restrictive assumption, the “correlation on average” between failures of the version could even be negative. This could even lead to zero expected system failure probability, even if the two version development processes were such that they each gave non-negligible probability of producing a program that fails on some demands ³. These two models (called *EL model* and *LM model* in what follows) both bring important insights:

- *even perfect isolation* – and thus independence – between the developments of the versions does not guarantee independence between their failures. Independent developments guarantee that, given a specific demand, two – independently “sampled” – versions will fail independently on that demand, and yet this in turn implies non-independence for a randomly chosen demand.
- furthermore, we get a clear formal description of conditions that increase failure dependence, and thus of which goals we should pursue when we try to “force” diversity.

These implications have been explored in many other applications of the same modelling approach, e.g. to the choice of fault removal methods [7], [8], to security [9], to human-machine systems [10].

Here, we focus on the consequences of dependence between development processes. In the EL and LM models, this issue does not arise: perfectly isolated development teams develop the programs for a multiple-version system. For brevity, we will call this the “independent sampling assumption”, or ISA. The ISA has two useful properties: it is mathematically simple enough to allow elegant theorems like the EL model’s implication of “positive correlation on average”, and it models the extreme ideal form of separation between developments of the versions. But there are many reasons for doubting that it will normally be realised in practice. Doubters point out, for instance, that

- some communication will tend to occur between the version development teams, at least indirectly;
- developers often share common education background or use the same reference books, etc.;

²This is often seen as an ideal condition. It would allow us to gain assurance of very high reliability of the redundant system at relatively low cost. For instance, we could trust that a 3-version parallel system has a probability of failure per demand (*pdf*) of no more than 10^{-6} at the rather affordable cost of demonstrating that each version has a *pdf* of no more than 10^{-2} . An even better scenario would be one in which common failures never happen, of course.

³There will always be *some* difference between the development processes of different versions, so the LM assumptions always apply when the development processes of the individual versions are independent. On the other hand, it is difficult (actually, there is no obvious method) to quantify how far the true conditions of development are from the EL, worst-case assumptions. So, if an assessor wishes to err on the side of conservatism, the EL assumptions are the only appropriate ones.

- the management of a multiple-version development will exert common influences on the development teams, e.g. by distributing clarifications and amendments to the specifications.

These scenarios prompt several questions: do they violate the ISA? If they do, do they invalidate the message from the Eckhardt and Lee breakthrough, that failure independence is unlikely and it is prudent to assume positive correlation? And do they invalidate any other practical guidance drawn from these models? In conclusion, *what are the practical implications of possible statistical dependencies* between the development (sampling) of versions? Answering these questions is the topic of this paper. To do so, we have to clarify how the relevant aspects of real-world processes are mapped into modelling assumptions.

We will conclude that we see no scenario that cannot be modelled via the ISA; yet, as we shall show, *the same real-world system development process may be correctly modelled as respecting or violating this assumption*, depending on the level of knowledge assumed about the history of a specific project.

In the process of this analysis, a further important question arises: is the ISA really an “extreme optimistic” assumption for the EL special case, which is what gives the EL result its value as a warning: “even under the most optimistic assumptions – perfect independence in development – still you should expect identical processes to produce positive correlation of failures”. So far, authors who recommend separation of version developments have plausibly argued that this would prevent the propagation of mistakes between the teams developers of different versions (“*fault leaks*” [1]). It is plausible that this propagation may occur, via either the direct imitation of erroneous solutions or the sharing of similar viewpoints and strategies (e.g. high-level architectural decisions) which frame the development problems similarly for the different teams, creating similar “blind spots” or error-prone subtasks. But the probability of common failures of two versions is the result of two factors: the correlation of the failures and the absolute probability of each one failing; results about independence or lack thereof say nothing about this second factor. If we accept that some events may occur that produce statistical dependency between the versions’ (otherwise independent) developments, why would these events not also change – perhaps improve – the average reliability produced by each version’s development process? For instance, consider the act of distributing a specification correction to the developers of multiple versions: the project managers are accepting reduced separation – more dependence – between developments because they seek greater reliability for each version. Other authors, and practitioners responsible for safety-critical systems, indeed maintain [11] that strict separation of developments, which they accept from previous literature such as [1] as necessary for effective diversity, will impede communications within a system development team to the *detriment* of system dependability. If the ISA turns out not to be a “best case” assumption, what changes when one releases it? Does the pessimistic warning from the EL model become invalid? And is the common advice to keep developments as “independent” (separate) as possible justifiable on mathematical grounds (using commonly accepted, empirically justifiable assumptions) or should it be judged on empirical grounds only?

Section II introduces the reference scenario for the discussion and some terminology, mostly adopted from the previous literature [6], [5], [12]. Section III recalls the previous theory (EL and LM models) so that in Section IV we can formally introduce the ISA. Section V shows ways of modelling violations of the ISA, and clarifies how whether the ISA holds depends on how we characterise the sample space for a given real-world scenario, not on the scenario itself. In Section VI we then discuss general theorems indicating preferences between system development processes; The discussion in Section VII deals with the practical implications of these mathematical considerations, either for decisions about managing a multiple-version development process or for any necessary changes in the lessons drawn from the EL/LM models. Finally, we summarise our general conclusions and indicate directions for future work.

II. REFERENCE SCENARIO AND TERMINOLOGY

Our reference scenario is a system that may be implemented either as a single version or as a diverse 2-channel, 1-out-of-2 system (Fig. 1). This is a very simple scenario, yet with practical applications (in safety systems) and presenting the essential difficulties of evaluating a probability of common failure.

We use the term “versions” (or “program versions”) in the sense of diverse, equivalent implementations of the system functions. Following common usage, when there is no risk of ambiguity, we will also call “version” a channel of the two-version system. We will avoid the other common meaning of the term “versions” to designate the results of successive changes to a program, or “releases”.

We refer to the following simple picture of multiple-version development: separate “version development teams”, each producing one version (and possibly further divided into sub-teams for design, coding, inspection, testing, etc). One “project management team” or “manager” defines the requirement specifications that the development teams must implement and the constraints under which they have to work, handles specification updates and decides on final acceptance of the developed versions. In our terminology the way each version is developed is a *version development process*, and we call *system development process* the combination of the development processes for the two or more versions in a system, plus the way they are coordinated.

We consider an “on demand” system. It receives a demand from the environment and the result of processing it is either a

success (a correct response) or a failure (an incorrect response) by the system. ⁴

The dependability measure of interest is the system's *probability of failure on demand (pfd)*. For our purposes the nature of the required response to a demand – e.g., whether it is turning on an alarm signal or controlling complex mechanical actuators – is irrelevant. We only distinguish between two types of response to a demand – success, i.e. correct behaviour and failure. Also, we only consider failures due to design faults, i.e., failures not covered by the usual analyses methods for “random” failures.

When executing the software, there is uncertainty about which demand it will next receive from its environment. This uncertainty can be described by a *demand profile*, i.e. an assignment of probabilities to every possible demand, x , in the *demand space*, \mathcal{X} . The demand profile summarises, and depends on, the circumstances in which the system is used (and will generally be known with some degree of imprecision). This is a distribution, $P(X = x)$, with respect to a random variable, X , defined over the space \mathcal{X} . We will use the phrase “a randomly chosen demand”, meaning a demand that occurs according to this random process. We follow the convention of using uppercase letters (X), for random variables or outcomes of random processes, and lowercase letters (x) for the values (numbers or vectors or names) which they can take. In all the analyses that follow, the demand profile is assumed as fixed and given, i.e., the system's *pfd* is the probability of the system failing on a demand X , randomly chosen according to this demand profile.

Each version may contain faults, determined by the uncertain and variable process of software development. Due to these faults, it fails deterministically on certain demands, its *failure set*. The sum of the probabilities of all these demands is the *pfd* of that version. So, the demand profile associates to the failure set of a version a specific value of *pfd*.

Referring to Fig. 2, a version fails when subjected to a demand that is part of its “failure set”, determined by mistakes in development. Independence between failures of the two versions would mean that the *pfd* associated to the intersection of the two versions' failure sets is exactly equal to the product of the probabilities associated to each of the two failures sets. There is no obvious reason why this should be so. Furthermore, the same pair of versions could be employed under different demand profiles. It would seem extraordinary that all possible demand profiles maintained the invariant of failure independence. As an extreme case, for a pair of versions, the two failure sets might be disjoint, giving zero common *pfd*. Or they might be identical, or one contained in the other.

III. EARLY CONCEPTUAL MODELS OF DEVELOPMENT AND FAILURE OF DIVERSE SYSTEMS

In this section we briefly recall the EL and LM models [5], [6].

A. Description of failure behaviour given full knowledge of programs

Given a program that behaves deterministically, i.e. for each demand it either deterministically processes it correctly or deterministically fails to process it correctly, we can define a Boolean *score function* $\omega(\pi, x)$, which is defined, for each demand x and given program or system π , as:

$$\omega(\pi, x) = \begin{cases} 0, & \text{if } \pi \text{ processes } x \text{ correctly} \\ 1, & \text{if } \pi \text{ fails on } x \end{cases}$$

Although the complete score function of a program or system is usually unknown, it is a useful device for reliability modelling. Successful correction of faults can be modelled by changes in the score function, for some demands, from 1 to 0.

If we choose a demand X at random (according to the given demand profile) and look at the score function of π on this demand, $\omega(\pi, X)$, $\omega(\pi, X)$ is itself a random variable, and the system's *pfd* is its expected value:

$$\begin{aligned} pfd &= P(\pi \text{ fails on } X) \\ &= E_X(\omega(\pi, X)) \\ &= \sum_{x \in \mathcal{X}} \omega(\pi, x) P(X = x) \end{aligned} \tag{1}$$

where the notation $E_X(\omega(\pi, X))$ designates the expected value, or mean, of the random variable $\omega(\pi, X)$, with respect to the distribution of the random variable X .

⁴We will deal exclusively with software that can be analysed in terms of discrete demands. A demand can be as simple as a single invocation of a re-entrant procedure, or as complex as the complete sequence of inputs to a flight control systems from the moment it is turned on before take-off to when it is turned off again after landing. The same approach and insights can be extended to modelling the probability of failure as a function of continuous time in continuously operating software, but the added model complexity is not worth introducing for the purpose of this discussion [13], [12].

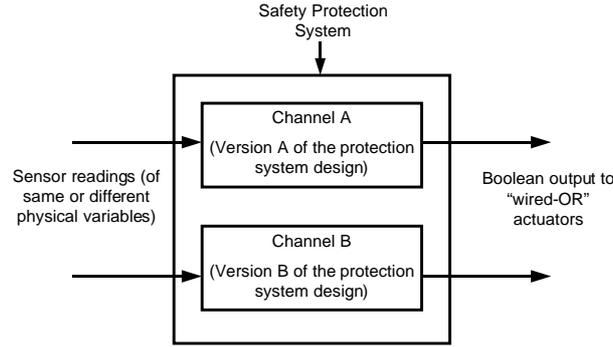


Fig. 1. Our reference system is an abstraction of a plant safety protection system (e.g. for a nuclear power plant) with two redundant, diverse channels: a simple 1-out-of-2 system. The system has to recognise a *demand* (a potentially hazardous state of the plant); the successful response to a demand is for the system to initiate a plant shut-down procedure. The output of each channel and of the whole system is thus logically a Boolean variable.

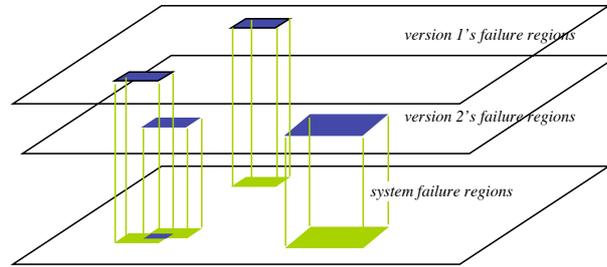


Fig. 2. Example of overlaps between the failure sets of two diverse software versions. The horizontal, rectangular surfaces each represent the complete set of system demands. The projection on the highest surface depicts those demands on which version 1 fails. The projection on the second highest surface depicts those demands on which version 2 fails. The overlaps of these projections, depicted in the lowest surface, shows those set of demands for which a 1-out-of-2 system, built from versions 1 and 2, will fail in operation.

B. Score function of a two-version system

If we consider a two-version, 1-out-of-2 system, its score function is given by the product of the score functions of the two program versions the system is made up of. Indeed, the system's score is 1 (failure) if and only if both versions' scores are also 1 (both fail).

If we call the two specific program versions in a system π_1 and π_2 , the *pdf* of the system they form is:

$$\begin{aligned}
 & P(\pi_1 \text{ and } \pi_2 \text{ fail on } X) \\
 = & E_X(\omega(\pi_1, X)\omega(\pi_2, X)) \\
 = & \sum_{x \in \mathcal{X}} \omega(\pi_1, x)\omega(\pi_2, x)P(X = x) \\
 = & pdf_1 pdf_2 + Cov_X(\omega(\pi_1, X), \omega(\pi_2, X))
 \end{aligned} \tag{2}$$

where the sign of the covariance term $Cov_X(\omega(\pi_1, X), \omega(\pi_2, X))$ captures the nature (positive or negative) of the failure correlation between the program versions π_1 and π_2 .

C. Difficulty function

We call “difficulty function” (the term introduced in [6] to name a concept initially formulated by Eckhardt and Lee [5]) the probability of a “randomly” developed software version failing on a particular demand. This program is randomly chosen from the population of different programs that (at least hypothetically) can be written to the same specification.

Formally, we require a set, $\mathcal{P} = \{\pi_i\}_{i \in I \subseteq \mathbb{N}}$ ⁵ of all possible program versions. We can reasonably assume that this set is finite. Any program must fit in some form of computer memory, whose size, at any stage of technology, is finite. Given a

⁵ \mathbb{N} is the set of natural numbers.

maximum feasible memory size L , we can assume as the set of all possible programs the set of all possible series of L zeros and ones. Of course, many of these “programs” will have zero probability of being produced. For a given development process and team we can define a random variable Π , whose realisation is a single version, e.g. π_i , $i \in I \subseteq \mathbb{N}$. We call *version sampling distribution* the distribution of Π , $P(\Pi = \pi_i)$, i.e., the function representing the probability of the event “the program version actually produced is program π_i ”.

This distribution is determined by the circumstances surrounding the development of the software: the specification of the program to be used, the members of the development team, the methods used in developing the software (including the verification and validation policy), the schedule and budget constraints, etc.

This way of modelling matches what we know about the variability of the outcome of a software development processes. Although the process is closely controlled, we know that its result – the software produced, including, importantly, the faults it contains – is not strictly determined by it. For instance, an assessor who is given all the documentation about the software’s development (usually showing, among other things, no evidence of residual faults in the delivered software) still does not know on which demands, if any, the software may fail due to unknown faults, although he/she may have an approximate idea of the quality to be expected from the software.

We can expect different “values” of the circumstances of development to induce different distributions of the random variable, Π . Given a particular development scenario (with its particular circumstances) and thus a distribution of Π , $\omega(\pi_i, x)$ then represents the score function of a specific program version, π_i , on demand x .

So, developing a program under given circumstances can indeed be described as running this stochastic production process once, or equivalently “extracting the program at random” from a population of many different programs, with their different probabilities $P(\Pi = \pi_1)$, $P(\Pi = \pi_2)$, \dots . We will use phrases like “a randomly selected program [version]” in this sense: the program may have been delivered, but it is still unknown in that its score function is unknown. Each possible version, π_i , has an associated score function, $\omega(\pi_i, x)$. The difficulty function on a particular demand x is then:

$$\theta(x) = E_{\Pi}(\omega(\Pi, x)) = \sum_{i \in I} \omega(\pi_i, x) P(\Pi = \pi_i). \quad (3)$$

D. Application to two-version system

We can now derive the first result of the EL and LM models: for two software versions, independently developed by two teams, it is inappropriate to estimate their joint *pdf* by multiplying their individual *pdf* estimates. The reasoning is as follows.

For a specific system development project (with its particular constraints and circumstances), the two redundant channels, A and B, have associated random variables, Π_A and Π_B . These represent the program version that will eventually be produced for each channel. Π_A and Π_B have associated version sampling distributions, e.g. $P(\Pi_A = \pi)$ represents the probability of team A, in charge of developing the program version for channel A, delivering the specific program version π . These version sampling distributions induce difficulty functions, $\theta_A(x)$ and $\theta_B(x)$, for each demand x . The EL model assumes that the same constraints (circumstances of development imposed on the development teams) cause the version development teams to develop their versions roughly “in the same way”, to the extent of “selecting them randomly” according to the same distribution, though rigorously independently. That is, $P(\Pi_A = \pi_i) = P(\Pi_B = \pi_i)$ for all π_i . Thus, for any given demand, the version development teams are equally likely to make mistakes on that demand: they have identical difficulty functions, i.e., $\theta_A(x) = \theta_B(x) = \theta(x)$ for every demand x . It can then be shown (see Appendix II) that, submitting to this randomly chosen pair of versions the same, randomly chosen, demand the probability of both failing is:

$$\begin{aligned} & P(\Pi_A, \Pi_B \text{ fail on randomly chosen } X) \\ = & P(\Pi_A \text{ fails on } X)P(\Pi_B \text{ fails on } X) + Var_X(\theta(X)) \\ = & (P(\Pi_A \text{ fails on } X))^2 + Var_X(\theta(X)) \\ = & (E_X[\theta(X)])^2 + Var_X(\theta(X)). \end{aligned} \quad (4)$$

where $Var_X(\theta(X))$ designates the variance of the random variable $\theta(X)$ and we have made use of the fact that for identically distributed version sampling processes we have $P(\Pi_A \text{ fails on } X) = P(\Pi_B \text{ fails on } X)$. Since the variance of any random variable is non-negative (4) shows that the average system *pdf* can be no better than the value $(E_X[\theta(X)])^2$, the value one would expect if the versions failed independently, on average. The LM model, however, recognises that constraints on the teams and, as a result, on the development of the two versions are unlikely to be identical: in general, $\theta_A(x) \neq \theta_B(x)$ for some x . In particular, the management team can *impose* different constraints on the two developments and sets of developers (“forced diversity”). So, unlike in EL,

$$\begin{aligned}
& P(\Pi_A, \Pi_B \text{ fail on randomly chosen } X) \\
&= P(\Pi_A \text{ fails on } X)P(\Pi_B \text{ fails on } X) \\
&\quad + Cov_X(\theta_A(X), \theta_B(X)) \\
&= (E_X[\theta_A(X)])(E_X[\theta_B(X)]) \\
&\quad + Cov_X(\theta_A(X), \theta_B(X)). \tag{5}
\end{aligned}$$

Possibly $Cov_X(\theta_A(X), \theta_B(X)) \leq 0$ so that the average system *pdf* could be better than $(E_X[\theta_A(X)])(E_X[\theta_B(X)])$; the lower bound for the mean system *pdf* is no longer the product term. In particular, the average system *pdf* could be 0 even if $E_X[\theta_A(X)]E_X[\theta_B(X)] > 0$.

IV. THE INDEPENDENT SAMPLING ASSUMPTION (ISA): IMPLICATIONS AND RELAXATION

The Independent Sampling Assumption of the EL and LM models is a plausible representation for the ideal of complete separation between the developments of the two versions: with “perfect” separation, there is no way that the development of one version may influence the development of another one.

The ISA implies *conditional independence*, given the specific demand x , between the failures of the two versions: for any given demand, the probability of that demand being a failure point for one version does not depend on whether it is a failure point for the other version. This in turn implies $\theta_{AB}(x) = \theta_A(x)\theta_B(x)$: the probability of building a two-version system that fails on x is the product of the probabilities of each version development team building a version that fails on x . Thus the ISA allows one to derive equations (4) and (5) (see Appendices II-A and II-B).

There are, however, several reasons for studying scenarios in which the ISA is false:

- complete separation is impossible for various practical reasons. So, we ought to study the effects of the inevitable, though possibly small, departures from it;
- communication between the teams may in some cases be desirable because:
 - either it causes positive correlation between failures on each demand, but improves the reliabilities of the individual versions so much that the net effect is improved system dependability,
 - or perhaps it can be engineered to cause negative correlation in such a way as to improve system dependability;
- even without communication between teams, the management may wish to improve the expected *pdf* of the diverse system by enforcing methods that plausibly violate conditional independence. Examples, as we shall see, are:
 - for the choice of algorithms to be implemented, allowing the two teams to choose freely but with the constraint that they use different algorithms for the same subset of the demand space. The hope is to produce negative correlation between the team’s mistakes on the same demand;
 - regarding quality assurance measures, mandating some common procedure which may cause *positive* correlation. For instance, testing the two versions on the same test cases may be a cost-effective way of improving the reliability of both versions created, improving the reliability of the fault-tolerant system.
- more subtly, we will show that the ISA is equivalent to assuming that the version sampling distributions incorporates somewhat complete knowledge of the circumstances of development. But to answer some important questions, we may need to model scenarios in which some of the circumstances are unknowns, i.e. random variables. This turns out to violate the ISA. Examples of such uncertain circumstances in development could be e.g. unforeseen deviations of the time and funds available for specific tasks from the pre-set project calendar and budget.

V. MODELLING DEPENDENCE BETWEEN VERSION DEVELOPMENTS

The development process is complex and, to some extent, random. Many factors affect the outcome of the development process. In addition, multiple version development processes may have factors in common. As a consequence, similarity of failures between the program versions may occur. Some examples of common factors include:

- similarities between the backgrounds of the development teams;
- specification errors or ambiguities or late changes, making specific sets of demands more “difficult” for both teams;
- communication between the teams, either direct (discussions of design problems, common project reviews) or indirect (queries to the project management causing specification clarifications to be issued to both teams);
- use of common design solutions or of common test suites.

All such possible sources of dependence between the teams in a development project, like many other events/circumstances in this process, can be modelled as random variables. We shall refer to these random events as [random] *influences*. That is, they are events/circumstances which affect the developments of the two versions and whose value may not be known beforehand. The simpler interpretation of this “randomness” is as “uncertainty in the world”: we are trying to predict the effects of a

process that has yet to happen and is affected by random factors. However, it can just as naturally represent “uncertainty in knowledge”: the development has taken place, but what we know about it is limited; we still do not know the values of all these variables.

We model an influence in development as a random variable, say E , which can take values from a set, \mathcal{E} , associated with a probability distribution describing the probability $P(E = e)$ of E taking the value $e \in \mathcal{E}$ during development. In our notation we will assume the set \mathcal{E} to be finite or countable to help intuition, though assuming E to be a continuous variable would require minimal changes.

At this abstract level, it does not matter whether an influence represents an event external to the software development process (e.g., a change in the requirements on which both version developments depend) or generated internally (e.g., selection of common tests for both versions) or even interactions between the teams (the specific information exchanged in both directions can be represented as a random variable or set of random variables). We will mostly be interested in the effects of *common* influences: those that affect the developments of both versions. We will show that such common influences may indeed increase correlation between version failures, *or* they may reduce it.

In producing a version, each team is somehow affected by the value taken by each influence: that is, this value affects each team’s version sampling distribution. For instance, an unforeseen reduction of the time available for V&V may increase the probability of less reliable versions being delivered. To describe this effect, we can express a team’s version sampling distribution in terms of conditional distributions, dependent on the values of the influences. For example, team A’s probability of delivering a certain version π_i , given a certain value e of V&V time is $P(\Pi_A = \pi_i | E = e)$ (written sometimes, for the sake of brevity, as $P(\Pi_A = \pi_i | e)$). Every distinct combination of values of the influences, i.e., different set of constraints on version development, determines a distinct version sampling distribution and thus difficulty function. As long as the values of the influences are unknown, the probability of a specific version being produced, and thus the difficulty function on any specific demand, are given by averaging these values over all possible combinations of values of the influences.

We can describe scenarios involving influences via Bayesian networks (or “Bayesian belief networks”, BBNs), as in Fig. 3 and Fig. 4 [14], [15], [16].⁶ These Bayesian networks depict hypothetical system development processes in which there may be multiple influences, common to the two versions as well as separate. The meaning of the graphs is as follows: each node is a random variable; the nodes without common parents are mutually independent random variables; the nodes with common parents are mutually independent, *conditionally* on the values of the parent nodes. For each node, a conditional probability distribution is defined: the distribution of the random variable associated with that node, conditional on all the values of the random variables associated with the node’s parents. The composition of all these distributions defines the joint distributions of all the random variables represented by all the nodes in the graph. To determine whether a system fails on a demand all that matters is the choice of the versions actually delivered for operation and of a demand, represented by the three nodes Π_A , Π_B and X in the graph. This is shown in the graphs by the fact that these are the only parent nodes of the nodes “ Π_A fails” and “ Π_B fails” which in turn are the only parents for “System fails”.

We note that nodes that are ancestors of only the Π_A or only the Π_B nodes do not represent *common* influences. They can be eliminated from a BBN model by averaging with respect to them the conditional probability distributions associated to their child nodes. Fig. 4 gives a BBN with examples of only common influences that we could be concerned about in an actual two-version development process.

The EL and LM models are represented by a BBN as in Fig. 5, which only contains the right-hand part of the BBNs in either Fig. 3 or Fig. 4. For any scenario of dependence in system development – for any one of our BBNs – *if all common influences are given specific values rather than chosen randomly OR if the version development processes have no common events (they are perfectly isolated from each other)*, then the two final versions are chosen independently, and the ISA (EL or LM model) applies.

We can transform Fig. 3 or Fig. 4 into a form without events in the development process that do not represent common influences, such as Fig. 6. This is accomplished by averaging (marginalising) over those nodes (random variables) in Fig. 3 that do not appear in Fig. 6. This transformation preserves the meaning of the original BBN – Fig. 3 or Fig. 4 – in an important sense: it implies no conditional independence assumptions that were not already implied by the original BBN; and all the joint distributions between the nodes in Fig. 6 are unchanged from those between the homologous nodes in the original BBNs [14]. However, Fig. 6 omits the details of how the influences affect the development process (e.g. which phase of development they affected).⁷

So, scenarios with quite different common influences, even if these influences affect different phases of development (e.g., errors in specification *vs* choices of the same system test cases for both version development processes), can be reduced to a common mathematical form from the viewpoint of dependence relations. We can thus formulate a set of theorems that

⁶It is mathematically possible to describe joint distributions of random variables that cannot be described by any Bayesian network, but we have found none, and we conjecture that none exists, that has practical interest for our topic.

⁷In the BBN in Fig. 6 the common influences are mutually independent, a property that we will shortly exploit. If Fig. 3 had sets of non-mutually independent common influences, to achieve this property in 6 we would first merge each such set into a single random variable (a “vector” random variable, whose set of possible values is the Cartesian product of the sets of possible values of all the random variables thus “merged”). The most general case of 6 would have a single common parent for the nodes Π_A , Π_B , representing “all common influences affecting the two developments”.

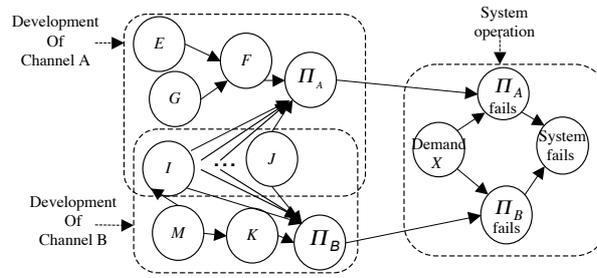


Fig. 3. This graph is a Bayesian network (BBN) depicting a (two-version) system development process affected by multiple influences, some of them common to the two versions. The nodes to the left of Π_A and Π_B might represent, for instance, specific design artefacts, test techniques and test cases selected, and influences on these various aspects of development like communication between the teams and the project management. The influences may interact in complex ways, e.g., mistakes in a specification document may affect choices of test cases and both affect which version is delivered. We have added the rounded boxes to identify the three main subsets of the BBN corresponding to the processes of developing the two versions and of operating the system.

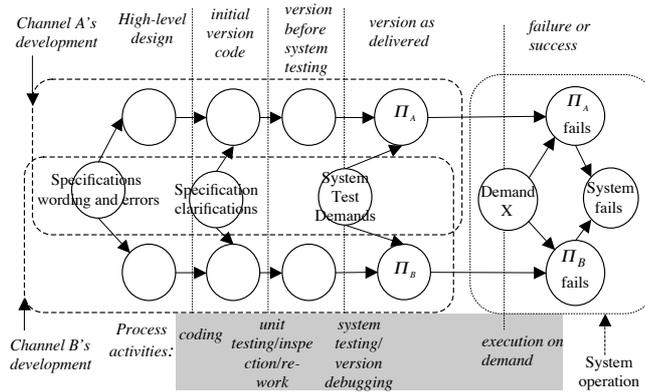


Fig. 4. Here we have represented some possible examples of common influences between the developments of two versions, which exert effects at various stages of the development processes and invalidate the Independent Sampling Assumption. The nodes in the top and bottom row represent artefacts at successive stages of production of the two versions. To avoid cluttering the diagram, the names of the artefacts are listed above it instead of giving a name to each node. Under the diagrams we have named the activities that transform an artefact into the next one. Each one is subject to some degree of randomness in its results, justifying the representation of each artefact as a random variable, whose distribution is determined by the exact values of its parent nodes: the artefact upstream of it, and in most stages a common influence as well, represented by a node in the middle row. In this case, only the unit testing phase is performed by each team in isolation and without common influences affecting both teams.

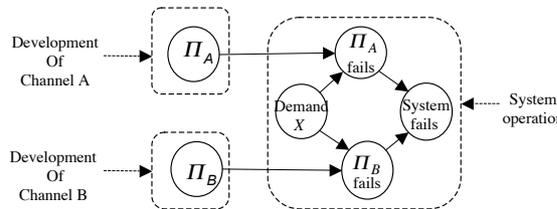


Fig. 5. A Bayesian network (BBN) for the EI and LM models. The two versions are chosen independently (ISA), which is represented by the absence of common parent nodes for Π_A and Π_B , and they fail independently *conditionally* on the randomly chosen demand X , as shown by the presence of the single common ancestor X for the two nodes “ Π_A fails”, “ Π_B fails”.

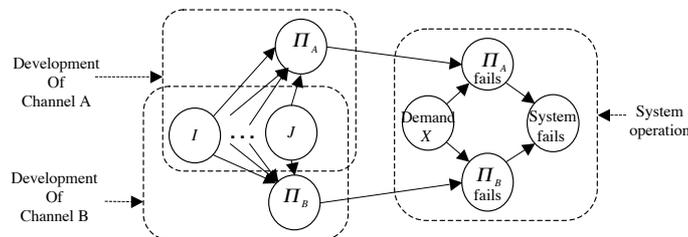


Fig. 6. The Bayesian network in Fig. 3, and all those we have examined that represent interesting scenarios of development, can be transformed into a shape like this one, in which the only influences are direct “parent” nodes of Π_A and Π_B , and are common to both version development processes. The intermediate nodes through which the influences affect Π_A and Π_B have been removed by marginalising the distributions in the BBN of Fig. 3 with respect to the non-common influences. Thus, all these scenarios can be described by the same form of equation that applies to this figure. We study this form of equation in this paper.

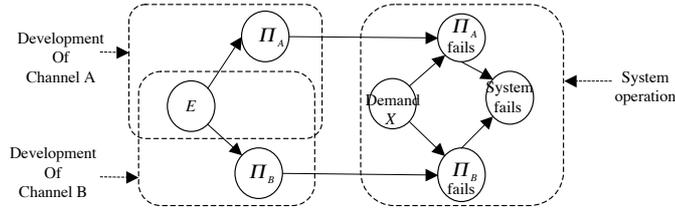


Fig. 7. Bayesian network for a system with one common influence between the development processes: e.g., the same test suite is used to test both versions.

depend only on the presence of common influences. All the scenarios of dependence in system development that we have considered so far in our study, some resulting in more complex BBNs than Fig. 3, can be transformed in this way, resulting in the applicability of the same theorems to seemingly different scenarios.

We next discuss the implications of these models. The mathematical details and theorem derivations used in this section are in Appendices I and III.

A. Effect of a single common influence

For the sake of simplicity, we first assume two version development processes sharing a single common influence E as in Fig. 7. If we first consider a specific instantiation of the two processes (including the specific value - say e - taken by E), leading to the production of a specific pair of versions, the value e of E determines for each one of the two versions a conditional version sampling distribution, $P(\Pi_A = \pi_i | e)$ and $P(\Pi_B = \pi_j | e)$, and difficulty function, which we call $\theta_{A,e}(x)$ and $\theta_{B,e}(x)$. $\theta_{A,e}(x)$ and $\theta_{B,e}(x)$ are functions of the demand, x , alone because they are evaluated given the specific value e of the random variable E . Since E is the only common influence, and its value is known to be e , nothing has really changed compared to the LM model in Sec. III, except that we know the two version sampling distributions to be the particular distributions that hold when $E = e$. In particular, for two arbitrary programs, π_i and π_j , $P(\Pi_A = \pi_i, \Pi_B = \pi_j | e) = P(\Pi_A = \pi_i | e)P(\Pi_B = \pi_j | e)$.

Let us compare this with a situation in which only the distribution of E is known, rather than its actual value: we do not know how the factor E manifests itself during the specific development project considered, but only the *a priori* constraints on the development process. We can show that this violates the ISA:

$$\begin{aligned} & P(\Pi_A = \pi_i, \Pi_B = \pi_j) \\ = & P(\Pi_A = \pi_j)P(\Pi_B = \pi_j) \\ & + Cov_E(P(\Pi_A = \pi_i | E), P(\Pi_B = \pi_j | E)) \end{aligned}$$

In this case we are interested in the difficulty functions as functions of the demand alone, taking into account the fact that E may take any value. We then need to average with respect to the influence E :

$$\begin{aligned} \theta_A(x) &= P(\Pi_A \text{ fails on a demand } x, \\ &\quad \text{given uncertainty about which value } E \text{ takes}) \\ &= E_{\Pi_A, E}[\omega(\Pi_A, x)] \\ &= \sum_{i \in I, e \in \mathcal{E}} \omega(\Pi_A, x) P(\Pi_A = \pi_i | e) P(E = e) \end{aligned}$$

The probability of a randomly chosen pair of versions failing together on a demand x , if E may take a random value, is:

$$\begin{aligned} \theta_{AB}(x) &= P(\Pi_A \text{ and } \Pi_B \text{ fail on demand } x, \\ &\quad \text{given uncertainty about which value } E \text{ takes}) \\ &= \theta_A(x)\theta_B(x) + Cov_E(\theta_{A,E}(x), \theta_{B,E}(x)) \end{aligned} \quad (6)$$

As a result, the expected system *pdf*(joint failure probability) has the form:

$$\begin{aligned} E_X(\theta_{AB}(X)) &= P(\Pi_A \text{ fails on } X) P(\Pi_B \text{ fails on } X) \\ &\quad + Cov_X(\theta_A(X), \theta_B(X)) \\ &\quad + E_X(Cov_E(\theta_{A,E}(X), \theta_{B,E}(X))) \end{aligned} \quad (7)$$

which, compared to the expression for the LM model, Eq. (5), has an additional covariance term. The covariance term in equation (6) indicates that in this view the failures of the two versions, even conditional on a given demand, are not independent.

That is, $\theta_{AB}(x) \neq \theta_A(x)\theta_B(x)$; equality (a zero covariance term) would hold if the version development processes A and B were independent. Equality would also hold if for one of the versions, e.g. A, the difficulty were a constant with respect to E , i.e., $\theta_{A;\epsilon}(x, \epsilon) = \theta_A(x)$; or, in other words, under this condition the factor E can be considered as not being an “influence” on version A, since it does not affect its version sampling distribution.

We can now see that there are different possible viewpoints on the same process, depending on which events or factors (“influences”), among the potentially variable circumstances in the development, we assume as fixed (as fully known or as fully determined by events that have “already happened”). The appropriate viewpoint depends on which questions we wish to answer. Each viewpoint implies its specific version sampling distributions, and thus difficulty functions, for the two teams.

The EL and LM models simply assume fixed values for any common influences that might be present in the system development process (more precisely, for those with respect to which the two difficulty functions, for the same demand, have non-zero covariance; the others can be ignored).⁸ In particular, the EL model assumes that all influences have been instantiated to some definite values: by the end of development, it is certainly so and the main result “with identical version development processes, your expectation of the probability of joint failures should be greater than the product of your expected *pdfs* for the two versions” is thus a sound warning. We will soon show examples of using “non-ISA” descriptions to derive other general conclusions. So, the fact that some viewpoints imply the Independent Sampling Assumption and some do not is just a mathematical curiosity, with an aspect of mathematical convenience when the ISA does apply. If we refer back to the “fault leak links” between version developments, discussed in [1], we can now see that some of them could usefully be represented in the difficulty function without violating the ISA, and others could not, but this always depends to some extent on the observation point that we choose. On considering the effects of a common influence with uncertain value we will usually need to assume the ISA is violated. If, on the other hand, we are considering the effects of a particular value of the influence then the ISA is not violated.

For instance, consider the concern that two development teams are likely to have had similar technical education, and thus share some preferred solutions for typical problems, leading to similar “typical” errors, which may tend to cause failures on the same demands. If we are considering two specific, existing teams, their educational backgrounds are determined, their typical errors are described in their respective difficulty functions, and thus education is not a “common” influence in our sense, because it is not a random factor. The similarity of backgrounds does not violate the ISA. The likelihood that it causes common failures is fully described by the two difficulty functions, through a contribution to their covariance over the space of demands, in equation 5.

Suppose now that the teams have yet to be selected; for each version, each possible team will determine a different difficulty function. If the selections for the two teams are affected by a common random factor, then the ISA is violated, creating the covariance term in equation 6. If this term is non-zero when averaged over the demands (last term in equation 7), the violation of the ISA invalidates the conclusions of the EL/LM models. A positive last term in equation 7 would mean, roughly, that when we choose a development team with high propensity to fail on certain demands, we become more inclined to choose a second team with similar weak spots on those demands. If instead no common factor affects the two team selections in this way, the uncertainty on which team will be chosen for each version can be factored into the difficulty function for that version (by averaging, on each demand, among all the difficulty functions, corresponding to every possible team).

B. The general case: multiple influences

Consider multiple, mutually independent influences, as in Fig.8: some common influences E_1, \dots, E_N , and others, I_A and I_B , each affecting just one of the two developments. It can be shown (see Appendix IV) that the previous equations generalise as follows. If we focus on one influence, e.g. E_1 , we see that

$$\begin{aligned}
 P(\Pi_A = \pi_i, \Pi_B = \pi_j) = & \\
 \sum_{i_A, i_B, e_2, \dots, e_N} & \left(P(\Pi_A = \pi_i | i_A, e_2, \dots \right. \\
 & \left. \dots e_N) P(\Pi_B = \pi_j | i_B, e_2, \dots e_N) \right. \\
 + Cov_{E_1} & \left[P(\Pi_A = \pi_i | E_1, \dots e_N, i_A), P(\Pi_B = \pi_j | E_1, \dots \right. \\
 & \left. \dots e_N, i_B) \right] \Big) P(I_A = i_A, I_B = i_B) P(E_2 = e_2, \dots \\
 & \dots E_N = e_N) \tag{8}
 \end{aligned}$$

⁸Going to an extreme viewpoint, we could assume as known every detail of the two developments, down to the two specific program versions created, say π_A and π_B : the difficulty functions $\theta_A(x)$ and $\theta_B(x)$ collapse to the “score functions” $\omega(\pi_A, x)$ and $\omega(\pi_B, x)$, and independence conditional on each demand is guaranteed by the fact that the score functions can only take the values 0 or 1.

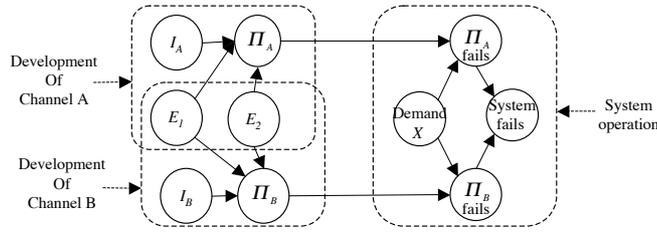


Fig. 8. This is a specific example of a BBN like the one in Fig. 3, but with some non-common influences.

where

$$P(\Pi_A = \pi_i | i_A, e_2, \dots, e_N) \\ = E_{E_1} \left[P(\Pi_A = \pi_i | E_1, \dots, e_N, i_A) \right]$$

and

$$P(\Pi_B = \pi_j | i_B, e_2, \dots, e_N) \\ = E_{E_1} \left[P(\Pi_B = \pi_j | E_1, \dots, e_N, i_B) \right]$$

So, each common influence induces correlation, captured by a covariance term, between the two version development processes. This implies that the joint difficulty function, $\theta_{AB}(x)$, no longer exhibits independence as we shall see below. The joint difficulty function can be rewritten (as detailed in Appendix IV) to emphasize the effect of a specific influence, e.g. E_1 , as:

$$\theta_{AB}(x) = \\ \sum_{e_2, \dots, e_N} \left(\theta_{A; e_2, \dots, e_N}(x) \theta_{B; e_2, \dots, e_N}(x) + \right. \\ \left. Cov_{E_1} \left[\theta_{A; E_1, \dots, e_N}(x), \theta_{B; E_1, \dots, e_N}(x) \right] \right) P(E_2 = e_2) \dots \\ \dots P(E_N = e_N) \quad (9)$$

where, say, $\theta_{A; e_2, \dots, e_N}(x) = E_{E_1, I_A}(\theta_{A; E_1, e_2, \dots, e_N, I_A}(x))$, i.e., it is the mean probability of failure on x given fixed values of E_2, \dots, E_N and averaged over all possible values of E_1 . $\theta_{B; e_2, \dots, e_N}(x)$ is similarly defined.

It is possible to expand the right hand side of the last equation (Please see Appendix IV). Thus, we obtain the joint difficulty function as

$$\theta_{AB}(x) = \\ P(\Pi_A \text{ fails on } x) P(\Pi_B \text{ fails on } x) + \\ Cov_{E_2, \dots, E_N} \left[\theta_{A; E_2, \dots, E_N}(x), \theta_{B; E_2, \dots, E_N}(x) \right] + \\ E_{E_2, \dots, E_N} \left[Cov_{E_1} \left[\theta_{A; E_1, \dots, E_N}(x), \theta_{B; E_1, \dots, E_N}(x) \right] \right] \quad (10)$$

Similar to equation (6) the joint difficulty function in equation (10) does not exhibit independence. From equation (10) we may obtain the expected probability of common failure (i.e. expected *pdf* of the 1-out-of-2 system) by averaging $\theta_{AB}(x)$ over all possible system demands. The right hand side of (10) thus becomes

- a product of the expected *pdfs* of the versions, plus
- a series of terms that are averages of covariances.

That is, the expected *pdf* of the 1-out-of-2 system is

$$\theta_{AB}(x) = \sum_{e_2, \dots, e_N} \left(\theta_{A; e_2, \dots, e_N}(x)^2 + \left(\text{Var}_{E_1} \left[\theta_{A; E_1, e_2, \dots, e_N}(x) \right] \right) \right) P(E_2 = e_2) \dots \dots P(E_N = e_N) \quad (12)$$

where $\text{Var}_{E_1} \left[\theta_{A; E_1, e_2, \dots, e_N}(x) \right] \geq 0$ always, since it is a variance. This is a natural consequence of the processes being identical. A practical example of this is discussed in [17] where both teams are required to test their versions using the same test suite, E_1 , randomly chosen from the space of all test suites, \mathcal{E} , according to a specified test suite generation method, $P(E_1 = e_1)$. This influence adds a variance term, as shown in equation (12), to the expression for average system *pdf*. If we replace this single random test suite with two independent, identically distributed random test suites E_{1A} and E_{1B} , one for each version development process, then this variance term disappears: the average system *pdf* is no worse than it would be with the single influence E_1 .

Observing that in equation (12) each common influence contributes a variance term, we can state a general rule as follows.

Preference criterion 1: Decoupling of mirrored version development processes. Irrespective of how many common influences exist between two mirrored version development processes, substituting a common influence with two influences (one for each of the two versions) with the same distribution as the one removed, but mutually independent, yields better system *pdf* (equal *pdf* as a limiting case).

Note that:

- after applying this “decoupling” with respect to a common influence, the resulting processes are still mirrored, and thus this criterion still applies: “decoupling” with respect to *any number* of common influences is an improvement;
- simply removing a common influence does not guarantee improvement. For instance, one way of removing the common influence in this example is to eliminate testing, which would change the two version sampling distributions, presumably for the worse, and may well therefore make the two-version system also worse. Instead, “decoupling” as defined above is beneficial because it does not change the two version sampling distributions but only (and for the better) their joint distribution.

B. Non-mirrored version development processes

If the version development processes are *not* mirrored, each common influence produces a covariance term in the expression of the joint difficulty function (equation 6) or more generally (10). In principle, this term can be positive or negative.

In particular, let us consider the case of a positive covariance term. We may state a general result as follows. Observe that in the equations above: (i) there is no preference about the order in which we choose the influences to “eliminate” from the outer summation; (ii) for each covariance term introduced, the successive transformations in the equation contain averages of that term over an increasing number of random variables (influences). Now, suppose there is a common influence E such that the covariance term introduced by E is known to be positive, i.e. it is known that – roughly – the values of the influence in question that imply worse probability of failure for one version imply worse probability of failure for the other version as well; suppose that this property holds for any possible combination of values of the other influences. Then, even after being averaged over all other influences, that covariance term is sure to remain a positive contribution to the system *pdf*. Likewise, if the covariance term introduced by common influence E is always negative, this term will remain negative even after averaging. In particular,

Preference criterion 2: Decoupling of diverse version development processes. Given two version development processes such that with respect to some common influence E the covariance of their difficulty functions is positive for any possible combination of values of all the other influences, substituting the common influence E with two influences (one for each of the two versions) with the same distribution as E , but mutually independent, yields better system *pdf*.

A special case of positive covariance is the one in which E is a numerical random variable and the two difficulty functions are monotonic functions (both non-increasing or both non-decreasing) of E (see Appendix V).

As an example, consider this scenario: Our two-version system is to control a new model of some kind of equipment, and part of its planned V&V process will be system testing in the equipment prototype. A certain time budget has been allocated for this system testing phase, but the actual time available may vary depending on when the prototype is actually ready. Both teams are thus exposed to the same random deviations: “time available for testing on the prototype” is a common influence.

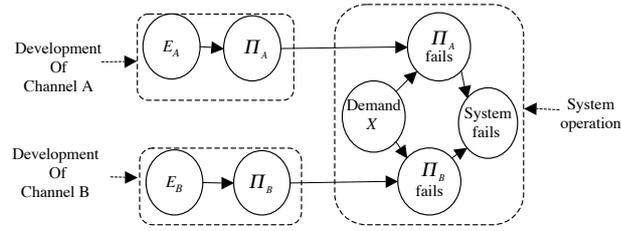


Fig. 9. Here, the system development process is as in Fig. 7, except for the fact that the two teams choose their test suites independently. The version sampling distributions and difficulty functions for the two channels are the same as in Fig. 7, but the process in this figure produces better system *pdf* (equal *pdf* as a limiting case). The random selection of demands is the only source of uncertainty in common between the channels.

We may reasonably expect an increase in the time available, allowing more testing, to improve the “difficulty” for every demand, irrespective of other conditions, and a decrease to make it worse. The teams’ difficulty functions can be viewed, therefore, as monotonically decreasing functions of the common influence. This is sufficient for the covariance term in equation (6) to be positive. This means that the uncertainty on the actual available time in fact adds to the probability of common failure of the average pair of versions developed, as compared to what would be calculated by just assuming the average effect of this variable influence. Although one will never in practice be able to calculate these expected *pdfs* from the equations, this is not a purely theoretical result. It tells us that if we had a choice between purchasing versions that had been originally affected by such a common influence (but without knowing its value: how much system testing time was actually available) and others that were developed independently – the development processes being otherwise *statistically* equal in the two scenarios – we should expect better system *pdf* from the latter.¹¹

Of course a dual theorem applies for negative covariance: substituting two non-common, identically distributed influences with a common one, with the same distribution, and such that the covariance with respect to it is negative, will improve the resulting system. Creating forms of negative covariance this way – that is, introducing random influences that whenever they happen to hamper one version development team on some demands, help the other team on the same demands – is useful in software development. By doing this we achieve increased levels of reliability at the cost of producing a single random influence. This generalises a “more reliability at no extra economic cost” result that was first discussed in [17]. There it was noted that in the case of testing it is hard to imagine strategies giving such negative covariance.

C. Homogeneous versus diverse development processes

Finally, let us consider again two mirrored version development processes, with a common influence E . A plausible way of “increasing diversity” is to require one of the two development teams to change its process in some aspect (change the probability distributions associated to some node in the BBN), so as to obtain a process that is intuitively different but promises to be no worse than the previous process. The difference might be e.g. in the technology used, or the testing methods, or in the algorithms implemented. Let us indicate the two processes as α and β . Suppose that α and β are equivalent in that they satisfy the “indifference” condition introduced in [6]: a system made of two versions, $\Pi_{A\alpha}$ and $\Pi_{B\alpha}$, produced by α has the same expected *pdf* as one of two versions, $\Pi_{A\beta}$ and $\Pi_{B\beta}$, produced by β :

$$E_{X, \Pi_{A\alpha}, \Pi_{B\alpha}, E} (\omega(\Pi_{A\alpha}, X)\omega(\Pi_{B\alpha}, X)) = E_{X, \Pi_{A\beta}, \Pi_{B\beta}, E} (\omega(\Pi_{A\beta}, X)\omega(\Pi_{B\beta}, X))$$

, where $E_{X, \Pi_{A\alpha}, \Pi_{B\alpha}, E} (\omega(\Pi_{A\alpha}, X)\omega(\Pi_{B\alpha}, X))$ and $E_{X, \Pi_{A\beta}, \Pi_{B\beta}, E} (\omega(\Pi_{A\beta}, X)\omega(\Pi_{B\beta}, X))$ are the average *pdfs* for the α and β based systems, respectively. Using the Cauchy-Schwarz inequality¹² and indifference as follows

$$\leq \frac{E_{X, \Pi_{A\alpha}, \Pi_{B\beta}, E} (\omega(\Pi_{A\alpha}, X)\omega(\Pi_{B\beta}, X))}{\sqrt{E_{X, \Pi_{A\beta}, \Pi_{B\beta}, E} (\omega(\Pi_{A\beta}, X)\omega(\Pi_{B\beta}, X))}}$$

¹¹We wondered whether one could identify some management regime that would neutralise this unwanted effect from such common variations in resources: e.g., would it be sufficient to require that if the two teams have to cut down on testing time, they must limit the cuts to the testing they perform separately, and to two disjoint subsets of the demand space? We have found that this assumption alone could not be proved to give an improvement; further assumptions would be needed, abstruse enough – in our opinion – that one would know whether to believe them in a specific project. Or could one just, ideally, isolate one of the two teams from a common influence? For instance, consider the common influence due to the possibility that effort may need to be diverted from the project, and in some companies these unplanned cuts would be likely to affect both development teams. Suppose that the management decides that any such cut will only affect team A, keeping team B’s plans intact. Then, with respect to this influence team A’s difficulty function would be constant, eliminating the corresponding positive covariance term in the r.h.s. of equation (6). But this is not guaranteed to be a net gain: team B may now be subject to the possibility of greater effort cuts than if team A had not been so privileged, so its average difficulty has increased and with it other terms on the r.h.s. of equation (6). Hence, this way of shielding one of the teams from the common influence, may or may not improve the expected system *pdf*.

¹²Let $f(x)$ and $g(x)$ be any two real, integrable functions then the Cauchy-Schwarz inequality states that $E[f(X)g(X)] \leq (E[f(X)^2]E[g(X)^2])^{1/2}$

$$\begin{aligned}
& \times \sqrt{E_{X, \Pi_{A\alpha}, \Pi_{B\alpha}, E} (\omega(\Pi_{A\alpha}, X)\omega(\Pi_{B\alpha}, X))} \\
= & E_{X, \Pi_{A\beta}, \Pi_{B\beta}, E} (\omega(\Pi_{A\beta}, X)\omega(\Pi_{B\beta}, X))
\end{aligned} \tag{13}$$

we conclude that

$$\begin{aligned}
& E_{X, \Pi_{A\alpha}, \Pi_{B\beta}, E} (\omega(\Pi_{A\alpha}, X)\omega(\Pi_{B\beta}, X)) \\
\leq & E_{X, \Pi_{A\beta}, \Pi_{B\beta}, E} (\omega(\Pi_{A\beta}, X)\omega(\Pi_{B\beta}, X)) .
\end{aligned} \tag{14}$$

Again, the result extends to version development processes with any number of common influences. So,

Preference criterion 3: Diversification between version developments. If two-version systems produced using a process α for both versions or using another process β for both versions yield the same expected system *pdf*, using process α for one version and process β for the other yields expected system *pdf* which is at least as good or better. In particular,

This generalises the similar result in [6], obtained under the ISA (independent version development processes). In [17] similar generalised results were obtained for a more restrictive problem description (the choice of a test suite for fault removal). Note that without the indifference assumption, this clear-cut preference does not hold.

A peculiarity of this ‘‘criterion for improvement’’ is a form of irreversibility. Once we alter two mirrored version development processes by ‘‘diversifying’’ some part of them, and thus turning variance terms in our equations into covariance terms, even adding a common influence will not undo the diversification, in the sense that it will not reintroduce variance terms into the expression for the average system *pdf*. I.e., adding this additional ‘‘coupling’’ may make the system less reliable, on average, as we have seen, but will not produce in the equations the feature that is a sufficient condition for this decreased reliability. Mathematically, once two version development processes are stochastically ‘‘diverse’’ (i.e., non-mirrored), adding common influences cannot make them ‘‘non-diverse’’ again.

VII. DISCUSSION

The questions we have studied concern the effects of project management policies on the system *pdf* of 1-out-of-N systems. For instance: between two multiple-version development processes that appear intuitively sound, which one should be picked? Or, given a process that we trust and a proposed change to this process, can we forecast whether the change will be an improvement?

Such decisions are characterised by great uncertainty because the system development process does not fully determine the resulting *pdf*: we need to reason in terms of probabilities and probability distributions. Besides, we normally lack much statistical evidence to suggest such probabilities. But rigorous probabilistic reasoning can at least clarify whether the assumptions that one believes to be true do support a specific decision. So, we look for characteristics of alternative development processes that, when modelled mathematically, allow us to choose one of the alternatives as the one to be preferred. We thus also implicitly outline the set of questions that cannot be answered via mathematics alone plus known empirical laws, and for which decisions must wait for new experimental evidence or depend on subjective opinion.

A. Mathematical results

One of our successes has been to relax some of the stringent assumptions of the previous theory based on the seminal work by Eckhardt and Lee and Littlewood and Miller (EL and LM models). Until now, most interesting theorems depended on very limiting assumptions about the real-world scenario modelled: typically, independent development of the versions, delivering versions with identical mean *pdf*. We have developed a natural, very general and powerful extension of these models.

An aspect of this generality is that practically any development process that we can imagine and any factor affecting system failure can be represented in our style of models, via the abstraction of ‘‘common influences’’, and all can be reduced to a common representation as in Fig. 4 and Section V-B. This includes all the factors that are usually considered in the literature as possible causes of similarity or correlation between the version development processes: from communication between the teams to common backgrounds among the developers. All such ‘‘influences’’ influencing the version development processes are modelled simply as random variables. The version development processes change as functions of the values taken by these variables.

Note that the ‘‘common influences’’ need not be limited to the development phase. For instance, if we are interested in physical failures as well as in software-caused failures, our BBNs can include extra nodes that are parents of the nodes ‘‘ Π_A fails’’ and ‘‘ Π_B fails’’, to represent any common stress factors like ambient temperature or common shocks.

Our models, which are quite abstract in that in most cases the functions that they refer to cannot be estimated in practice, clarify the relationship between the intuitive ideas of ‘‘separation’’, ‘‘independence’’ and ‘‘diversity of process’’, formal concepts of independence and correlation, and the measures of interest like reliability or probability of failure on demand. But these models also offer some direct practical help for decision making: we were able to derive three ‘‘preference criteria’’ among processes for developing two-version systems, based on sufficient conditions which, we think, people will recognise to match their assumptions in certain practical decision problems.

An example of the power of these models is that from our preference criteria one can derive, as special cases, the results about the selection of test suites for two versions, [17], and generalise them to any number of stages of testing: for any number of stages of testing, given equal development processes for the pre-test versions, separate, *independent generation of the test suites for the two versions is always better* (i.e., may improve mean system *pdf* but cannot make it worse) than choosing the same suite for the two versions.

Fig. 11 demonstrates how our results enlarge the set of scenarios in which mathematically founded preferences can be stated between alternate ways of running multiple-version development. Our “preference criteria” describe changes that improve the system development process by shifting it from one domain to another as depicted in Fig. 11.

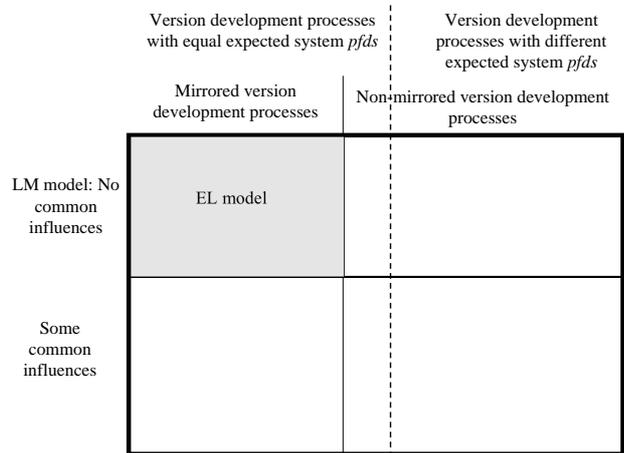


Fig. 10. The space of possible system development process states under consideration

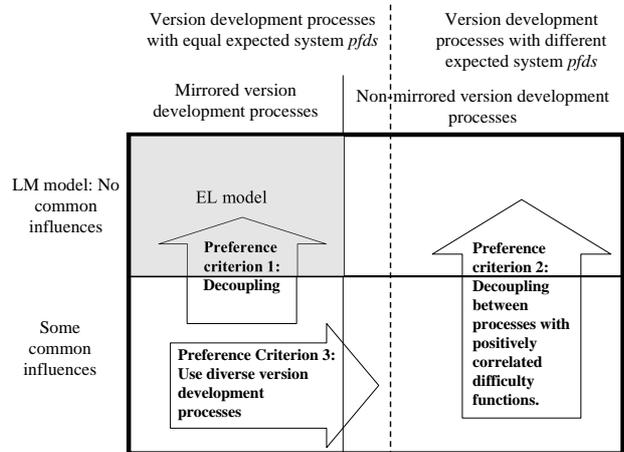


Fig. 11. The space of possible assumptions about a system development process, and the subsets on which the various results recalled or derived here apply. The arrows indicate our “preference criteria”: following an arrow from a subset of scenarios into another one improves the expected *pdf*. Note that the arrow on the right indicates an improvement in the positive covariance case of our preference criterion 2; in the negative covariance case, the arrow would be in the opposite direction.

In particular, an improvement over previous theory is in addressing questions such as, “When does combining multiple ways of ‘forcing’ diversity [3], [18] bring more benefit than simply ‘forcing’ diversity in one way?” The preference criteria outlined above give sufficient conditions for this question to be answered, for instance. An earlier theorem (section IV in [6]) showed that combining multiple such decisions is desirable, but under rather restrictive conditions of independent sampling of versions, complete “indifference” between alternatives, and of each decisions being a choice between mutually exclusive sets of possible programs.

Applying, as we did, the notation of Bayesian networks seems an improvement in itself, in that it visualizes relationships

of conditional dependence, and their implications (“given that this factor can affect the outcome of this stage of development, will these two random variables be independent?”) can be recognised by applying simple rules based on the graph’s topology (concerning e.g. the existence of common ancestor nodes).

Our results also clarify the meaning of “independence” between version developments, and somewhat reduce its importance. Whether two version developments are independent (in the probability theory sense of the word) depends largely on how much we assume to be uncertain about them. Independence holds, for all scenarios of real-world development, *once all the random factors affecting both versions have taken specific values*: after the last interaction between the development teams, for instance, their development processes have been altered by the specific communications exchanged (the specific *instantiation* of the random event “communication between the teams”) but from that moment onwards the past intercommunication between the two processes no longer affects their independence. From this viewpoint, the EL and LM results are generally valid. If, on the other hand, we wish to assess the expected results of the version development processes when the values of the random common influences are still unknown, then the two developments will not generally be independent. This latter viewpoint is the appropriate one for answering some interesting questions. For example, to understand the effect of adding a common influence at a certain stage of the version development processes, it helps to assume as determined and thus independent the processes *before* that stage, and see the added influence as what might violate this independence, with positive or negative effects.

B. Updates to accepted principles and opinions

A major question for us was whether the insight derived from the earlier “EL” and “LM” models [12] that assumed the version development processes to be strictly “independent” (in an intuitive sense that also has a stringent mathematical definition) remains valid, despite the fact that this condition is never fully guaranteed in practice, and at times it is even advisable to violate it intentionally. For instance, does the main message obtained from the EL result still hold, i.e., that a prudent assessor, given identical version development processes, should assume positive correlation between version failures? Indeed it does, because with mirrored version development processes the ISA was indeed the most optimistic assumption (cf. equation (12)), so that the conclusion applies *a fortiori* if it is violated.

Broadly speaking, looking at the generally accepted, intuitive idea that effective diversity depends on strong “separation” and “diversification” between the version development processes, our results by and large support them, at least for the simpler scenarios. Separation should be sought between “*mirrored*” version development processes, or system development processes where common influences would induce positive failure correlation between the system versions. However, our results also underscore how many subtle variations are possible with respect to these intuitive cases.

We could for instance – a possibly surprising result – have situations where the presence of a common influence reduces failure correlation: diverse version development processes might be made “even more diverse” via negative covariation between difficulty functions (equations (6) and (10)). In practical terms, this would support policies that dynamically alter the process applied to one version to make it “as different as possible”, from some viewpoint, from that applied to the other.

Another limitation to the support we can offer for the intuitive principles above is that our preference criteria depend on simple sufficient conditions (e.g. they support decisions about removing a common influence if it induces positive covariance between difficulty functions, irrespective of the values of other “influences”) and when these are not satisfied the criteria no longer help: to decide between alternative system development processes (i.e., BBN topologies) one must then look for empirical evidence.

Our general models make it easy to include in the descriptions of “cause of dependence” the cases of diversity-reducing influences that yet improve system *pdf* (e.g., testing with a common, randomly chosen test suite); as well as the possibility, at least in theory, of common influences that improve system *pdf* by *increasing* diversity. Dependence in the system development process is not in itself good or bad. What matters are questions like: does the common influence that “creates” the dependence affect the teams’ difficulty functions in “the same way”? Do both teams become more or less likely to make mistakes on given demands due to a change in an influence?

There is room for further results from this modelling approach. There may be other “preference criteria” with sufficient conditions that are clear enough to be recognised in practical situations. It would be especially interesting to find more examples of system development policies that should create useful, negative covariance between the difficulty functions of the two processes.

Another area for future work is extending our modelling beyond 1-out-of-2 systems. While these are an important category of systems, and illustrate the basic problems in managing “diversity”, results applicable to other fault-tolerant architectures would be desirable. Littlewood and Miller [6] showed that many results do not extend in intuitive ways from the 1-out-of-2 case, and further subtle, counter-intuitive effects are possible.

In conclusion, the advances described here greatly broaden the set of situations for which preferences between ways of developing diverse software can be stated on a purely mathematical basis. They give rigorous support to common-sense beliefs about the importance of separation and diversity between development processes in order to achieve failure diversity, but also rigorously delimit the premises under which these beliefs are justified. Also, they highlight some less intuitive results; for instance, in defining scenarios of “negative dependence” between version development processes or in showing how the

very existence of development factors, influencing both versions' reliabilities in a consistent way, may increase their average common *pdf*.

ACKNOWLEDGMENT

This work was partially supported by the DIRC project ('Interdisciplinary Research Collaboration in Dependability of Computer-Based Systems') funded by UK Engineering and Physical Sciences Research Council (EPSRC), and by the DISPO (DIverse Software PrOject) projects, funded by British Energy Generation Ltd and BNFL Magnox Generation under the Nuclear Research Programme via contracts PP/40030532 and PP/96523/MB.

REFERENCES

- [1] A. Avizienis, "The methodology of n-version programming," in *Software Fault Tolerance*, M. Lyu, Ed. John Wiley and Sons, 1995, pp. 23–46.
- [2] M. Lyu and Y. He, "Improving the n-version programming process through the evolution of a design paradigm," *IEEE Transactions on Reliability*, vol. R-42, pp. 179–189, 1993.
- [3] B. Littlewood and L. Strigini, "A discussion of practices for enhancing diversity in software designs," Centre for Software Reliability, City University, DISPO project technical report LS-DI-TR-04, 2000. [Online]. Available: http://www.csr.city.ac.uk/people/lorenzo.strigini/ls.papers/DISPO2000_diversityEnhancing/
- [4] J. C. Knight and N. G. Leveson, "An experimental evaluation of the assumption of independence in multi-version programming," *IEEE Transactions on Software Engineering*, vol. SE-12, pp. 96–109, 1986.
- [5] D. E. Eckhardt and L. D. Lee, "A theoretical basis for the analysis of multiversion software subject to coincident errors," *IEEE Transactions on Software Engineering*, vol. SE-11, pp. 1511–1517, 1985.
- [6] B. Littlewood and D. R. Miller, "Conceptual modelling of coincident failures in multi-version software," *IEEE Transactions on Software Engineering*, vol. SE-15, pp. 1596–1614, 1989.
- [7] B. Littlewood, P. Popov, L. Strigini, and N. Shryane, "Modelling the effects of combining diverse software fault removal techniques," *IEEE Transactions on Software Engineering*, vol. SE-26, pp. 1157–1167, 2000.
- [8] D. Bosio, B. Littlewood, M. J. Newby, and L. Strigini, "Advantages of open source processes for reliability: clarifying the issues," 2002. [Online]. Available: http://www.csr.city.ac.uk/people/lorenzo.strigini/ls.papers/402_open_source/
- [9] B. Littlewood and L. Strigini, "Redundancy and diversity in security," in *ESORICS 2004, 9th European Symposium on Research in Computer Security*, Series: Lecture Notes in Computer Science, P. Ryan and P. Samarati, Eds. Sophia Antipolis, France: Springer-Verlag, 2004, pp. 423–438.
- [10] L. Strigini, A. A. Povyakalo, and E. Alberdi, "Human-machine diversity in the use of computerised advisory systems: a case study," in *DSN 2003, International Conference on Dependable Systems and Networks*, San Francisco, U.S.A., 2003, pp. 249–258.
- [11] Y. C. B. Yeh, "Design considerations in boeing 777 fly-by-wire computers," in *3rd IEEE High-Assurance Systems Engineering Symposium (HASE)*. Washington, DC, USA: IEEE Computer Society Press, 1998, pp. 64–73.
- [12] B. Littlewood, P. Popov, and L. Strigini, "Modelling software design diversity - a review," *ACM Computing Surveys*, vol. 33, pp. 177–208, 2001.
- [13] P. T. Popov and L. Strigini, "Conceptual models for the reliability of diverse systems - new results," in *28th International Symposium on Fault-Tolerant Computing (FTCS-28)*. Munich, Germany: IEEE Computer Society Press, 1998, pp. 80–89.
- [14] S. Lauritzen, *Graphical Models*, Series: Oxford Statistical Science Series. Clarendon Press, Oxford, 1996, vol. 17.
- [15] P.-J. Courtois, B. Littlewood, L. Strigini, D. Wright, N. Fenton, and M. Neil, "Bayesian belief networks for safety assessment of computer-based systems," in *System Performance Evaluation: Methodologies and Applications*, E. Gelenbe, Ed. CRC Press, 2000, pp. 349–363.
- [16] CACM, "Real-world applications of bayesian networks," *Communication of the ACM, Special Issue*, vol. 38, no. 3, 1995, – SHIP T046.
- [17] P. Popov and B. Littlewood, "The effect of testing on the reliability of fault-tolerant software," in *DSN 2004, International Conference on Dependable Systems and Networks*. Florence, Italy: IEEE Computer Society, 2004, pp. 265–274.
- [18] P. Popov, L. Strigini, and A. Romanovsky, "Choosing effective methods for design diversity - how to progress from intuition to science," in *SAFECOMP '99, 18th International Conference on Computer Safety, Reliability and Security*, Series: Lecture Notes in Computer Science, M. Felici, K. Kanoun, and A. Pasquini, Eds. Toulouse, France: Springer, 1999, pp. 272–285.
- [19] L. Strigini, "A note on the limits of average predictions: demonstrating how difficulty functions hide information," Centre for Software Reliability, City University, DISPO2 Project Technical Report LS-DISPO2-02, 2003.
- [20] —, "Bounds on survival probabilities given an expected probability of failure per demand," Centre for Software Reliability, City University, DISPO2 Project Technical Report LS-DISPO2-03, 2003.

APPENDIX I

THE RELATIONSHIP BETWEEN THE DIFFICULTY FUNCTION, THE *pdf* AND THE EXPECTED *pdf* OF A SYSTEM

We deal with the following probabilities of failure:

- failure of a specific program π on a specific demand x : this is the score function $\omega(\pi, x)$ of that program;
- failure of a specific program π on a randomly chosen demand: the *pdf* of program π , $\phi(\pi)$;
- failure of a randomly chosen program version on a specific demand x : the difficulty function $\theta(x)$;
- failure of a randomly chosen program version on a randomly chosen demand.

These four are linked as follows. The difficulty function $\theta(x)$ is the expected value (for a randomly chosen program version) of the score for a specific demand, as in equation (3). The *pdf* of the particular version, π , can be expressed as the expected value of its score on a randomly selected input, X , according to (1):

$$\phi(\pi) = E_X[\omega(\pi, X)] = \sum_{x \in \mathcal{X}} \omega(\pi, x) P(X = x) \quad (15)$$

Finally, if we select at random both a program version and a demand, the probability of that program version failing on that demand is obtained by averaging over both all possible versions and all possible demands:

$$E_X[\Theta] = \sum_{x \in \mathcal{X}} \theta(x) P(X = x) = E_{\Pi}[\Phi] = \sum_{i \in I \subseteq \mathbb{N}} \phi(\pi_i) P(\Pi = \pi_i) \quad (16)$$

where $\Theta \equiv \theta(X)$ is a random variable representing the "difficulty" of a randomly selected input and $\Phi \equiv \phi(\Pi)$ is a random variable representing the *pdf* of a randomly chosen program version. $E_X(\Theta)$, equal to $E_\Pi(\Phi)$ is the expected value of the "difficulty" function or the *pdf* respectively. It may not represent the probability of failure or the *pdf* of any particular version; it is a characteristic of the development process as described by the version sampling distribution.

We should note that the mean *pdf* is not a complete description of the population of possible programs and systems. For discussion of the limits and possible extensions see [19], [20].

APPENDIX II

DERIVATION OF THE EXPECTED SYSTEM *pdf* UNDER THE EL AND LM MODELS

A. Expected system *pdf* in the EL model

In what follows we show the derivation of the expected system *pdf* under the assumptions of the EL model. This derivation was hinted at in section III-D of the main text. Under the EL model, since $P(\Pi_A = \pi_i) = P(\Pi_B = \pi_i)$ for all π_i , it follows that $\theta(x) = \theta_A(x) = \theta_B(x)$ for all x . Consequently,

$$\begin{aligned}
& P(\Pi_A, \Pi_B \text{ fail on a demand 'x'}) \\
= & \theta_{AB}(x) \\
= & \sum_{i \in I \subseteq \mathbb{N}, j \in J \subseteq \mathbb{N}} \omega_i(x) \omega_j(x) P(\Pi_A = \pi_i, \Pi_B = \pi_j) \\
= & \sum_{i \in I \subseteq \mathbb{N}, j \in J \subseteq \mathbb{N}} \omega_i(x) \omega_j(x) P(\Pi_A = \pi_i) P(\Pi_B = \pi_j) \\
= & \sum_{i \in I \subseteq \mathbb{N}} \omega_i(x) P(\Pi_A = \pi_i) \sum_{j \in J \subseteq \mathbb{N}} \omega_j(x) P(\Pi_B = \pi_j) \\
= & \theta_A(x) \theta_B(x) \\
= & \theta(x)^2
\end{aligned}$$

It then follows that

$$\begin{aligned}
& P(\Pi_A, \Pi_B \text{ fail on randomly chosen } X) \\
= & E_X(\theta_{AB}(X)) \\
= & E_X(\theta(X)^2) \\
= & E_X(\Theta^2) \\
= & (E_X[\Theta])^2 + Var_X(\Theta)
\end{aligned}$$

where $\Theta = \theta(X)$ is the random variable representing the "difficulty" functions, for each team, on a randomly chosen demand, X .

B. Expected system *pdf* in the LM model

Similarly, we shall derive the expected system *pdf* under the assumptions of the LM model. So

$$\begin{aligned}
& P(\Pi_A, \Pi_B \text{ fail on a demand 'x'}) \\
= & \theta_{AB}(x) \\
= & \sum_{i \in I \subseteq \mathbb{N}, j \in J \subseteq \mathbb{N}} \omega_i(x) \omega_j(x) P(\Pi_A = \pi_i, \Pi_B = \pi_j) \\
= & \sum_{i \in I \subseteq \mathbb{N}, j \in J \subseteq \mathbb{N}} \omega_i(x) \omega_j(x) P(\Pi_A = \pi_i) P(\Pi_B = \pi_j) \\
= & \sum_{i \in I \subseteq \mathbb{N}} \omega_i(x) P(\Pi_A = \pi_i) \sum_{j \in J \subseteq \mathbb{N}} \omega_j(x) P(\Pi_B = \pi_j) \\
= & \theta_A(x) \theta_B(x)
\end{aligned}$$

and thus

$$\begin{aligned}
& P(\Pi_A, \Pi_B \text{ fail on randomly chosen } X) \\
= & E_X(\theta_{AB}(X)) \\
= & E_X(\theta_A(X) \theta_B(X)) \\
= & E_X(\Theta_A \Theta_B) \\
= & (E_X[\Theta_A]) (E_X[\Theta_B]) \\
& + Cov_X(\Theta_A, \Theta_B)
\end{aligned}$$

so that

$$P(\Pi_A, \Pi_B \text{ fail on randomly chosen } X) = (E_X [\Theta_A]) (E_X [\Theta_B]) + Cov_X (\Theta_A, \Theta_B)$$

where $\Theta_A \equiv \theta_A(X)$ and $\Theta_B \equiv \theta_B(X)$ are the random variables representing the “difficulty” functions of the two methods, A and B, on a randomly chosen demand.

APPENDIX III EFFECTS OF A SINGLE COMMON INFLUENCE

Let us assume two version development processes sharing a single, common influence E as in Fig. 7. If we first consider a specific instantiation of the two processes, including the specific value - say e - taken by E , leading to the production of a specific pair of versions, the value e of E determines for each one of the two versions a conditional version sampling distribution, $P(\Pi_A = \pi_i|e)$ and $P(\Pi_B = \pi_j|e)$, and difficulty function, which we call $\theta_{A:e}(x)$ and $\theta_{B:e}(x)$. $\theta_{A:e}(x)$ and $\theta_{B:e}(x)$ are functions of the demand, x , alone because they are evaluated given the specific value e of the random variable E . These difficulty functions are computed in the “usual” way, e.g.:

$$\begin{aligned} & \theta_{A:e}(x) \\ = & P(\Pi_A \text{ fails on demand } x | E \text{ takes value } e) \\ = & E_{\Pi_A:e}[\omega(\Pi_A, x)] \\ = & \sum_{i \in I} \omega(\Pi_A, x) P(\Pi_A = \pi_i | e) \end{aligned}$$

Since E is the only common influence, and its value is known to be e , nothing has really changed compared to the LM model in Sec. III, except that we know the two version sampling distributions to be the particular distributions that hold when $E = e$. In particular, for two arbitrary programs, π_i and π_j , $P(\Pi_A = \pi_i, \Pi_B = \pi_j | e) = P(\Pi_A = \pi_i | e) P(\Pi_B = \pi_j | e)$. Using this independence we may claim:

$$\begin{aligned} & \theta_{AB:e}(x) \\ = & P(\Pi_A \text{ and } \Pi_B \text{ fail on demand } x | E \text{ takes value } e) \\ = & E_{\Pi_A, \Pi_B:e}[\omega(\Pi_A, x) \omega(\Pi_B, x)] \\ = & \sum_{i \in I, j \in J} \omega(\pi_i, x) \omega(\pi_j, x) P(\Pi_A = \pi_i, \Pi_B = \pi_j | e) \\ = & \sum_{i \in I, j \in J} \omega(\pi_i, x) \omega(\pi_j, x) P(\Pi_A = \pi_i | e) P(\Pi_B = \pi_j | e) \\ = & \theta_{A:e}(x) \theta_{B:e}(x). \end{aligned}$$

That is, the failures of the two versions on a given demand are independent, conditionally on the demand x . Let us compare this with a situation in which we do not know the specific value taken by E , but only its distribution: we do not know how the factor E manifests itself during a specific development, but only the *a priori* constraints on the development process. We can show that this violates the ISA:

$$\begin{aligned} & P(\Pi_A = \pi_i, \Pi_B = \pi_j) \\ = & \sum_{e \in \mathcal{E}} P(E = e) P(\Pi_A = \pi_i, \Pi_B = \pi_j | e) \\ = & Cov_E(P(\Pi_A = \pi_i | E), P(\Pi_B = \pi_j | E)) + \\ & \sum_{e \in \mathcal{E}} P(E = e) P(\Pi_A = \pi_i | e) \times \\ & \sum_{e \in \mathcal{E}} P(E = e) P(\Pi_B = \pi_j | e) \\ = & P(\Pi_A = \pi_i) P(\Pi_B = \pi_j) \\ & + Cov_E(P(\Pi_A = \pi_i | E), P(\Pi_B = \pi_j | E)) \end{aligned}$$

In this case we are interested in the difficulty functions as functions of the demand alone, taking into account the fact that E may take any value. We then need to average with respect to the influence E :

$$\begin{aligned}
& \theta_A(x) \\
= & P(\Pi_A \text{ fails on a demand } x, \\
& \text{given uncertainty about which value } E \text{ takes}) \\
= & E_{\Pi_A, E} [\omega(\Pi_A, x)] \\
= & \sum_{i \in I, e \in \mathcal{E}} \omega(\Pi_A, x) P(\Pi_A = \pi_i | e) P(E = e)
\end{aligned}$$

The probability of a randomly chosen pair of versions failing together on a demand x , for a random value of E , is:

$$\begin{aligned}
\theta_{AB}(x) &= P(\Pi_A \text{ and } \Pi_B \text{ fail on demand } x, \\
& \text{given uncertainty about which value } E \text{ takes}) \\
&= E_E \left(\theta_{AB, E}(x) \right) \\
&= E_E \left(\theta_{A, E}(x) \theta_{B, E}(x) \right) \\
&= \sum_{e \in \mathcal{E}} \theta_{A, e}(x) \theta_{B, e}(x) P(E = e) \\
&= \sum_{e \in \mathcal{E}} \left(\sum_{i \in I, j \in J} \left[\omega(\pi_i, x) \omega(\pi_j, x) P(\Pi_A = \pi_i | e) \times \right. \right. \\
& \quad \left. \left. P(\Pi_B = \pi_j | e) \right] \right) P(E = e) \\
&= \sum_{i \in I, j \in J} \omega(\pi_i, x) \omega(\pi_j, x) \left(P(\Pi_A = \pi_i) P(\Pi_B = \pi_j) \right. \\
& \quad \left. + Cov_E(P(\Pi_A = \pi_i | e), P(\Pi_B = \pi_j | e)) \right) \\
&= \left(\sum_{i \in I} \omega(\pi_i, x) P(\Pi_A = \pi_i) \right) \\
& \quad \times \left(\sum_{j \in J} \omega(\pi_j, x) P(\Pi_B = \pi_j) \right) + \\
& \quad \sum_{i \in I, j \in J} \left(\omega(\pi_i, x) \omega(\pi_j, x) \times \right. \\
& \quad \left. Cov_E(P(\Pi_A = \pi_i | e), P(\Pi_B = \pi_j | e)) \right) \\
&= \theta_A(x) \theta_B(x) + Cov_E(\theta_{A, E}(x), \theta_{B, E}(x))
\end{aligned}$$

So, the average system *pdf* is

$$\begin{aligned}
E_X(\theta_{AB}(X)) &= E_X(\theta_A(X)) E_X(\theta_B(x)) \\
& \quad + Cov_X(\theta_A(X), \theta_B(X)) \\
& \quad + E_X(Cov_E(\theta_{A, E}(X), \theta_{B, E}(X)))
\end{aligned}$$

By evaluating, in a different order, the expectations used in obtaining the last equation we may recast the average system *pdf* in its alternate form;

$$\begin{aligned}
E_{\Pi_A, \Pi_B}(\phi(\Pi_A, \Pi_B)) &= E_{\Pi_A}(\phi(\Pi_A)) E_{\Pi_B}(\phi(\Pi_B)) \\
& \quad + Cov_X(\theta_A(X), \theta_B(X)) \\
& \quad + \sum_{\pi_i, \pi_j} \left(Cov_E(P(\Pi_A = \pi_i | E), P(\Pi_B = \pi_j | E)) \right. \\
& \quad \left. \times Cov_X(\omega(\pi_i, X), \omega(\pi_j, X)) \right)
\end{aligned}$$

where $\phi(\Pi_A, \Pi_B) = \phi(\Pi_A)\phi(\Pi_B) + Cov_X(\omega(\Pi_A, X), \omega(\Pi_B, X))$.

APPENDIX IV EFFECTS OF MULTIPLE COMMON INFLUENCES

Following the discussion in section V-B we assume multiple influences, as in Fig.8 or Fig.3: some common, mutually independent influences E_1, \dots, E_N , and mutually independent influences affecting just one of the two developments. So the probability of building a system with versions π_i and π_j is

$$\begin{aligned}
P(\Pi_A = \pi_i, \Pi_B = \pi_j) &= \sum_{\substack{e_1, \dots, e_N, i_A, i_B \\ \dots E_N = e_N, I_A = i_A, I_B = i_B}} P(\Pi_A = \pi_i, \Pi_B = \pi_j | e_1, \dots, e_N, i_A, i_B) P(E_1 = e_1, \dots \\ &= \sum_{\substack{e_1, \dots, e_N, i_A, i_B \\ \dots E_N = e_N})} P(\Pi_A = \pi_i, \Pi_B = \pi_j | e_1, \dots, e_N, i_A, i_B) P(E_1 = e_1, \dots \\ &= \sum_{\substack{e_1, \dots, e_N, i_A, i_B \\ \dots E_N = e_N})} P(\Pi_A = \pi_i, \Pi_B = \pi_j | e_1, \dots, e_N, i_A, i_B) P(E_1 = e_1, \dots \\ &= \sum_{\substack{e_1, \dots, e_N, i_A, i_B \\ \dots e_N, i_B})} \left(P(\Pi_A = \pi_i | e_1, \dots, e_N, i_A) P(\Pi_B = \pi_j | e_1, \dots \right. \\ &= \sum_{\substack{e_1, \dots, e_N, i_A, i_B \\ \dots e_N, i_B})} \left(P(\Pi_A = \pi_i | e_1, \dots, e_N, i_A) P(\Pi_B = \pi_j | e_1, \dots \right. \\ &\quad \left. \left. P(E_1 = e_1) P(E_2 = e_2) \dots P(E_N = e_N) P(I_A = i_A, I_B = i_B) \right) \right)
\end{aligned}$$

where we have exploited the independence between Π_A and Π_B conditional on the influences, and the independence among all influences.

The general form of the right-hand side of this equation evolves as follows: in evaluating the product term within the summation,

In particular, this equation reduces to equation (8) upon focusing on one influence, E_1 say, from which we can see that every common influence induces correlation, captured by covariance terms, between the version development processes. This implies that the joint difficulty function, similar to equation (6), no longer exhibits independence. Its form is

$$\begin{aligned}
\theta_{AB}(x) &= \theta_A(x)\theta_B(x) \\ &+ \sum_{\pi_i, \pi_j} \left(\omega(\pi_i, x)\omega(\pi_j, x) E_{I_A, I_B} \left(Cov_{E_1, E_2, \dots, E_N} \left[P(\Pi_A = \pi_i | E_1, \dots \right. \right. \right. \\ &\quad \left. \left. \left. \dots E_N, I_A), P(\Pi_B = \pi_j | E_1, \dots, E_N, I_B) \right] \right) \right) \\ &= \theta_A(x)\theta_B(x) + \left(Cov_{E_1, E_2, \dots, E_N} \left[E_{I_A}(\theta_{A, E_1, \dots, E_N, I_A}(x)), E_{I_B}(\theta_{B, E_1, \dots \right. \right. \\ &\quad \left. \left. \dots E_N, I_A}(x)) \right] \right) \tag{17}
\end{aligned}$$

To isolate the effect of a certain influence, E_1 say, we can reorder the calculation of $\theta_{AB}(x)$ as:

$$\begin{aligned}
\theta_{AB}(x) &= \sum_{e_2, \dots, e_N} \left(P(E_2 = e_2) \dots P(E_N = e_N) \sum_{e_1} P(\Pi_A \text{ fails on } x | E_1 = e_1, E_2 = e_2, \dots \right. \\
&\quad \dots, E_N = e_N) P(\Pi_B \text{ fails on } x | E_1 = e_1, E_2 = e_2, \dots \\
&\quad \left. \dots, E_N = e_N) P(E_1 = e_1) \right) \\
&= \sum_{e_2, \dots, e_N} \left(\theta_{A;e_2, \dots, e_N}(x) \theta_{B;e_2, \dots, e_N}(x) + Cov_{E_1} \left[\theta_{A;E_1, \dots, e_N}(x), \theta_{B;E_1, \dots, e_N}(x) \right] \right) P(E_2 = e_2) \dots P(E_N = e_N) \\
&= \sum_{e_3, \dots, e_N} \left(\theta_{A;e_3, \dots, e_N}(x) \theta_{B;e_3, \dots, e_N}(x) + Cov_{E_2} \left[\theta_{A;E_2, \dots, e_N}(x), \theta_{B;E_2, \dots, e_N}(x) \right] \right) \\
&\quad + E_{E_2} \left(Cov_{E_1} \left[\theta_{A;E_1, \dots, e_N}(x), \theta_{B;E_1, \dots, e_N}(x) \right] \right) P(E_3 = e_3) \dots P(E_N = e_N) \\
&= \theta_A(x) \theta_B(x) + Cov_{E_2, \dots, E_N} \left[\theta_{A;E_2, \dots, E_N}(x), \theta_{B;E_2, \dots, E_N}(x) \right] \\
&\quad + E_{E_2, \dots, E_N} \left[Cov_{E_1} \left[\theta_{A;E_1, \dots, E_N}(x), \theta_{B;E_1, \dots, E_N}(x) \right] \right] \tag{18}
\end{aligned}$$

Therefore,

$$\begin{aligned}
E_X \left[\theta_{AB}(X) \right] &= E_X \left[\theta_A(X) \theta_B(X) \right] + E_X \left[Cov_{E_2, \dots, E_N} \left[\theta_{A;E_2, \dots, E_N}(X), \theta_{B;E_2, \dots, E_N}(X) \right] \right] + \\
&\quad E_X \left[E_{E_2, \dots, E_N} \left[Cov_{E_1} \left[\theta_{A;E_1, \dots, E_N}(X), \theta_{B;E_1, \dots, E_N}(X) \right] \right] \right]
\end{aligned}$$

or

$$\begin{aligned}
E_X \left[\theta_{AB}(X) \right] &= E_X \left[\theta_A(X) \right] E_X \left[\theta_B(X) \right] + Cov_X \left[\theta_A(X), \theta_B(X) \right] \\
&\quad + E_X \left[Cov_{E_1} \left[\theta_{A;E_1}(X), \theta_{B;E_1}(X) \right] \right] \\
&\quad + E_X \left[E_{E_1} \left[Cov_{E_2} \left[\theta_{A;E_1, E_2}(X), \theta_{B;E_1, E_2}(X) \right] \right] \right] \\
&\quad + \dots + E_X \left[E_{E_1} \left[\dots E_{E_N} \left[Cov_{E_N} \left[\theta_{A;E_1, E_2, \dots, E_N}(X), \theta_{B;E_1, E_2, \dots, E_N}(X) \right] \right] \right] \right] \tag{19}
\end{aligned}$$

where $\theta_{A;e_2, \dots, e_N}(x) = E_{E_1, I_A}(\theta_{A;E_1, e_2, \dots, e_N, I_A}(x))$, i.e., it is the mean probability of failure on x given set values of E_2, \dots, E_N but averaged with respect to E_1 and I_A .

APPENDIX V

PROOF OF NON-NEGATIVE CORRELATION BETWEEN DIFFICULTY FUNCTIONS THAT ARE MONOTONICALLY DECREASING FUNCTIONS OF A COMMON INLUENCE

In the expressions for expected system *pdf* discussed in the main text there are terms that are expectations of covariances. For instance, terms like $E_X(Cov_E(\theta_{A;E}(X), \theta_{B;E}(X)))$ appear in section V. We shall show that the covariance of two monotonically decreasing difficulty functions is non-negative. Consequently, an expectation of this sort of covariance results in a non-negative value.

Theorem 0.1:

Let f and g be functions of the Random Variable, Y . Let there exist a point $y = \bar{y}$, such that

$$\begin{cases} f(y) \geq E_Y(f(Y)) & : y \leq \bar{y}, \\ f(y) \leq E_Y(f(Y)) & : y > \bar{y} \end{cases}$$

so that $\int_{y \leq \bar{y}} (f(y) - E_Y(f(Y))) dF_Y(y) \geq 0$ (\int is the usual Lebesgue-Stieltjes integral with respect to $F_Y(y)$, the cumulative distribution function of Y) and let g be a monotonically decreasing, real-valued function of y . That is, g is a real-valued function such that $g(x_1) \leq g(x_2)$ whenever $x_1 \geq x_2$. Then

$$Cov_Y(f(Y), g(Y)) \geq 0$$

Proof: Consider that

$$\begin{aligned} 0 &= \int_y (f(y) - E_Y(f(Y))) dF_Y(y), \text{ from the definition of expectation} \\ &= \int_{y > \bar{y}} (f(y) - E_Y(f(Y))) dF_Y(y) + \int_{y \leq \bar{y}} (f(y) - E_Y(f(Y))) dF_Y(y), \text{ from the additive property of the} \\ &\quad \text{Lebesgue-Stieltjes integral} \\ &\Rightarrow - \int_{y \leq \bar{y}} (f(y) - E_Y(f(Y))) dF_Y(y) = \int_{y > \bar{y}} (f(y) - E_Y(f(Y))) dF_Y(y) \end{aligned}$$

Using this result, the requirements $\begin{cases} f(y) \geq E_Y(f(Y)) & : y \leq \bar{y}, \\ f(y) \leq E_Y(f(Y)) & : y > \bar{y} \end{cases}$ and g , a monotonically decreasing function of y , we see that

$$\begin{aligned} &\int_{y > \bar{y}} (f(y) - E_Y(f(Y))) g(y) dF_Y(y) \geq g(\bar{y}) \int_{y > \bar{y}} (f(y) - E_Y(f(Y))) dF_Y(y) \\ &= -g(\bar{y}) \int_{y \leq \bar{y}} (f(y) - E_Y(f(Y))) dF_Y(y) \geq - \int_{y \leq \bar{y}} (f(y) - E_Y(f(Y))) g(y) dF_Y(y) \\ &\Rightarrow \int_y (f(y) - E_Y(f(Y))) g(y) dF_Y(y) \geq 0 \end{aligned}$$

Recall: We may always write $Cov_Y(f(Y), g(Y)) = \int_y (f(y) - E_Y(f(Y))) g(y) dF_Y(y)$. Using this above gives the required result;

$$Cov_Y(f(Y), g(Y)) \geq 0$$

■

corollary 0.2: $E_X(Cov_E(\theta_{A;E}(X), \theta_{B;E}(X))) \geq 0$ whenever the difficulty functions, $\theta_{A;E}(x)$, $\theta_{B;E}(x)$, are monotonically decreasing functions of the influence E .

Proof:

Suppose $\theta_{A;E}(x)$, $\theta_{B;E}(x)$ and the random variable E have the properties of f , g and Y respectively, from the last theorem. Then

$$Cov_E(\theta_{A;E}(X), \theta_{B;E}(X)) \geq 0 \Rightarrow E_X(Cov_E(\theta_{A;E}(X), \theta_{B;E}(X))) \geq 0$$

■

Part Verif – APPENDIX

(Methods and Tools for Verifying Resilience)

Non-Determinism in Multi-Party Computation (Abstract)

Michael Backes*

Birgit Pfitzmann[†]

Michael Waidner[‡]

Abstract

Outside security, non-determinism is an important tool for specifying systems without fixing unnecessary details. In security, however, normal refinement of non-deterministic specifications is usually not applicable, in particular because it may invalidate secrecy properties. Especially simulatability-based security notions seem to require detailed deterministic or probabilistic specifications. We show how one can nevertheless use the reactive simulatability (RSIM) framework to address non-determinism. In particular we survey its *generic distributed scheduling* for treating the non-determinism of asynchronous execution, discuss the experiences we made with this, and how it encompasses other recent scheduling approaches. We also show how property-based specifications can play the role of highest-level non-determinism in the RSIM context, and how functional non-determinism of machines can be captured by the system-from-structure derivations as well as by call-outs to the adversary or more general resolvers.

1 Introduction

In normal design processes, non-determinism is an important tool for initially specifying systems without fixing unnecessary details. Outside security, there are many well-accepted notions of refinement of non-deterministic specifications, e.g., for program verification and distributed systems. In security, however, normal refinement is usually not applicable, in particular because it may invalidate secrecy properties. In cryptography, this is particularly visible in simulatability-based approaches at system specification and refinement, and it may even seem inherent that such specifications cannot be non-deterministic: In cryptographic simulatability definitions, ultimately the views of certain parties are compared in the sense of computational indistinguishability [9]. For this, the views must be families of probability distributions. Hence at this point in the definition, all non-determinism in the specification or implementation of the protocol must have been resolved deterministically or probabilistically. As an example, we sketch the general RSIM definition from [7] in Figure 1.

However, we will show that this is no fundamental problem for using non-determinism in designing cryptographic multi-party protocols, because there are ample opportunities to resolve initial non-determinism within the overall formula in which the view comparison occurs.

2 Scheduling – Non-Determinism by Asynchronous Execution

A particular question that has recently found renewed interest is how the inherent non-determinism in the execution order of asynchronous systems can be resolved in cryptographic multi-party computation.

*Saarland University, Saarbrücken, Germany, backes@cs.uni-sb.de

[†]IBM Zurich Research Lab, Switzerland, bpf@zurich.ibm.com

[‡]IBM Software Group, Somers, USA, wmi@us.ibm.com

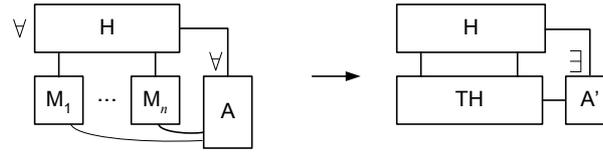


Figure 1: Overview of general reactive simulatability (RSIM). An implementation (often called real system) is on the left, the corresponding specification (ideal system) on the right. Here the views of H must be indistinguishable.

2.1 Typical Scheduling Patterns

In the distributed systems community, this resolution is usually done by a separate, arbitrary full-information scheduler, i.e., a component that at each step knows the entire system state and can base its next scheduling decision on that. For typical computational cryptographic systems, this gives the scheduler too much power. For instance, the scheduler can see internal secrets of honest parties and encode them in scheduling decisions that the adversary can learn [5]. In cryptography, the most typical scheduling pattern is therefore that the adversary schedules everything. However, in some cases even this scheduling is too strong.

- When *liveness*, *availability*, or *fairness* properties of a protocol are considered, some fairness of the underlying scheduling must usually be assumed, because certain messages have to reach their recipients. Cryptographic versions of such properties and corresponding schedulers were introduced in [3].
- *Covert channel prevention* is needed when the absence of information flow between certain parties is considered. Here an adversary should not be able to encode the information whose flow is otherwise prevented into scheduling information. Cryptographic versions of such definitions were introduced in [1].
- *Subprogram-like machine combination*. Proofs of distributed systems often use splitting and recombination of machines with properties such as associativity. Process algebras like π -calculus (first used cryptographically in [5]) have many such properties predefined, and also for the probabilistic IO automata (PIOA) model in the RSIM framework such properties were shown. If every machine recombination would make different channels external and thus open them to adversarial scheduling, it would significantly hinder such modular proofs. Hence it is useful to allow immediate local scheduling of certain channels [5, 8].
- *Adversary-scheduled secure channels*. Secure channels are sometimes needed in initial protocol phases such as the exchange of symmetric master keys. It seems realistic that even if an adversary cannot read and modify messages on such channels, it may be able to influence the channel speed. This is adversarial scheduling, but for channels where the adversary is neither the sender nor the recipient.

2.2 Generic Distributed Scheduling

The generic distributed scheduling from the RSIM framework [8] allows all the cases described above, alone or in combination, as well as many other scheduling mechanisms that one might come up with. All this is done with very little overhead compared with standard machine and scheduling definitions. The following two principles are used:

- Schedulers are normal machines.
- For each channel, one can designate which machine schedules it.

Thus the only addition to a concrete specification or a system definition, compared with a system model with fixed scheduling, is that for each channel, not only a sender and a recipient are designated, but also the scheduler. Clearly, all the cases from Section 2.1 can easily be defined as patterns in this model. For specifications or systems that use one of these patterns the scheduling can largely be given by reference to the pattern. (Clearly if, e.g., some channels are scheduled locally and others by the adversary, then one still has to designate which channels are which.) The fact that schedulers are normal machines also makes it easy to define one or many schedulers (e.g., a scheduler hierarchy or local schedulers), to provide each scheduler with arbitrary information, and to define arbitrarily how much an adversary learns from a scheduler (typically nothing beyond what it learns from other sources).

2.3 Discussion and Comparison of Scheduling Models

As shown above, special schedulers are usually needed in cryptography if adversarial scheduling is too strong. I.e., given the “normal” machines, a limited set of possible schedulers is defined, e.g., all fair ones that schedule certain channels. The overall set of behaviours of such a system is a subset of the behaviors that can occur with adversarial scheduling, because everything a separate scheduler and an adversary can do could easily be done by a combined adversary too. Thus every security *property* that can be proved for adversarial scheduling also holds for the restricted scheduling, but not vice versa.

Concerning *simulatability* definitions, specific scheduling patterns fall under the existing RSIM definitions and theorems (in particular composition) as long as all involved schedulers can be classified in the quantifier orders as either normal machines, adversaries, or honest users.¹ For all patterns above this is true. Hence generic distributed scheduling has been very useful for treating all these cases with only one set of definitions and theorems. If other quantifier orders are desired (quantifier orders are separate from the basic models in the RSIM framework and many variations have already been compared, starting with [6]), similar theorems need to be reproved. These proofs can follow the same graphical meta-structure as used for the RSIM proofs. For instance, we believe that a composition theorem for the quantifier order $\forall A \exists A' \forall H \forall S \exists S'$, where S and S' are the schedulers in the implementation and specification, (suggested by Robert Segala) can be proved without any serious change to the proof for general RSIM in [8].²

It is even possible with generic distributed scheduling to define full-information schedulers or super-polynomial schedulers for certain system parts (the former simply by letting the machines in this system part send their entire new state to their scheduler in each step), while keeping the adversary polynomial-time and with realistic information. Then more behaviors than with adversarial scheduling are possible. However, we do not believe that there are many cryptographically interesting uses of this: If the machines that are scheduled with full information contain secrets, in most cases the scheduling can leak these secrets to the adversary. If they do not contain secrets, the full-information scheduler cannot do much more than a normal adversary.

All other scheduling models proposed in the literature can, at least on this informal level, be easily mapped into generic distributed scheduling. Let us show this for a particular model [4] that was recently

¹The RSIM framework, in contrast to some related frameworks, allows quantification also over normal machines by considering systems consisting of many possible actual structures; Derivations of such a system from one “intended” structure are an additional definition layer that is currently mostly used for trust models, but can also be used for adding schedulers.

²Recall that for security *properties* the structure with separate S and A is *weaker* than the standard structure. However, if weaker structures on both sides are compared, there is no trivial relation to standard definitions.

built without any look at related literature (as the introduction to an earlier public version shows and several authors admitted); it can thus count as independent confirmation of the generality of the generic distributed scheduling from RSIM. Like the RSIM framework, it uses PIOAs. However, it schedules message output instead of message arrival. Hence the models are not easy to map formally, but neither the task PIOA authors we spoke to nor we currently think that this makes a significant difference. The “tasks” in that model correspond to the channels in the RSIM framework and in the cryptographic π -calculus from [5]: Like channels, these tasks group messages that the scheduler can schedule without knowing the exact message content. There is one specific scheduler (called “task schedule”), separate from the adversary. It does not get information from the running system and is deterministic. In the RSIM framework such a scheduler would be represented as a deterministic machine without normal in- and output channels. Note that this is a very weak scheduling model for security *properties*. Another variant with full-information schedulers for certain system parts, corresponding to the case we described above, is sketched.

3 Functional Non-Determinism in Simulatability Definitions

In design processes, functional non-determinism is rather more important than asynchrony. Typically, non-deterministic machines are used to leave certain choices open, which can be fixed by later refinement. If the design process for a system that contains cryptography is performed entirely with refinement steps that are proven correct with a simulatability definition, such a use of non-determinism may at first glance seem impossible because currently the basic machines in all such frameworks are probabilistic (which includes deterministic, but not non-deterministic).

Nevertheless, functional non-determinism can be handled in two ways. First, in the RSIM framework, one could easily allow non-determinism in the intended structures from which systems (sets of actual structures) are derived. Standard refinement notions could be used for this derivation. As the view comparison as in Figure 1 is only performed on the actual structures, it is not affected.

Secondly, one can call out potential non-deterministic choices and leave them to the adversary. In [7] (and the corresponding longer reports) this falls under the method for identifying “tolerable imperfections”. It was, e.g., used to leave open in a specification how many rounds a synchronous protocol takes, or to allow that the length of a message leaks partially or entirely. The same technique was explicitly or implicitly used by many subsequent protocol specifications used with simulatability definitions. One could generalize these techniques similar to the scheduling, i.e., generally introduce a class of components, say “resolvers”, that take choices left open by others. In the basic RSIM machine and execution model this simply makes no difference, but one could again come up with different quantifier orders between adversaries and other resolvers.

4 Non-Determinism by Property-based Specifications

The first specifications in a real-life design process are typically non-deterministic in a more fundamental way than non-deterministic machines: They consist of individual requirements. Ideally those are solicited from human stakeholders and then formalized in some logic, e.g., temporal logic for typical functional requirements on distributed systems. In security, additional requirements are secrecy requirements for certain data or message types. Such properties can also be formalized at an abstract level (one might call this “ideal properties”) and given a cryptographic semantics. It can be shown that the refinement of such properties by abstract system specifications (“ideal systems”) and later simulatability-based refinement to cryptographic systems are compatible. This was first formalized for integrity properties in [7]; secrecy properties exist in

more variants, e.g., [2].

5 Conclusion

Non-determinism is an important aspect of general system design processes. We have shown how, in spite of the probabilistic nature of typical cryptographic definitions, non-determinism can play its role also for systems containing cryptographic protocols. In particular the following four mechanisms can be used (roughly ordered from high-level to low-level specifications): cryptographic semantics for property specifications, system definitions via derivations from non-deterministic structures, call-outs of non-deterministic choices to the adversary, and generic distributed scheduling for the general low-level resolution of non-deterministic choices.

Acknowledgments. We thank Ling Cheung, Joshua Guttman, Nancy Lynch, John Mitchell, Roberto Segala, and Matthias Schunter for interesting discussions. This work is partially supported by the European Commission through the IST Programme under Contract IST-2002-507932-NOE ECRYPT. In this abstract we could not give a complete literature overview; we refer to the cited papers for more prior and related work.

References

- [1] M. Backes and B. Pfitzmann. Computational probabilistic non-interference. In *Proc. 7th ESORICS*, volume 2502 of *LNCS*, pages 1–23. Springer, 2002.
- [2] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing*, 2(2):109–123, 2005.
- [3] M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial fairness and liveness. In *Proc. 15th IEEE CSFW*, pages 160–174, 2002.
- [4] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Task-structured probabilistic I/O automata, 2006. To appear as Technical Report, RU Nijmegen, 2006. Preliminary version of April 2006 at Ling Cheung’s homepage.
- [5] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM CCS*, pages 112–121, 1998.
- [6] B. Pfitzmann, M. Schunter, and M. Waidner. Secure reactive systems. IBM Research Report RZ 3206, May 2000. http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz.
- [7] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM CCS*, pages 245–254, 2000.
- [8] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symp. on Security & Privacy*, pages 184–200, 2001. More model details in IACR ePrint Report 2004/082 with M. Backes.
- [9] A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd FOCS*, pages 80–91, 1982.

Soundness Limits of Dolev-Yao Models

Michael Backes¹, Birgit Pfizmann², and Michael Waidner³

¹ Saarland University, Saarbrücken, Germany, backes@cs.uni-sb.de

² IBM Research, Rueschlikon, Switzerland, bpf@zurich.ibm.com

³ IBM Software Group, Somers, USA, wmi@us.ibm.com

Abstract. Automated tools such as model checkers and theorem provers for the analysis of security protocols typically abstract from cryptography by Dolev-Yao models, i.e., they replace real cryptographic operations by term algebras. The soundness of Dolev-Yao models with respect to real cryptographic security definitions has received significant attention in the last years. Until recently, all published results were positive, i.e., they show that various classes of Dolev-Yao models are indeed sound with respect to various soundness definitions.

Here we discuss impossibility results. In particular, we present such results for Dolev-Yao models with hash functions, and for the strong security notion of blackbox reactive simulatability (BRSIM)/UC. We show that the impossibility even holds if no secrecy (only collision resistance) is required of the Dolev-Yao model of the hash function, or if probabilistic hashing is used, or certain plausible protocol restrictions are made. We also survey related results for XOR. In addition, we start to make some impossibility results explicit that tacitly underly prior soundness results in the sense of motivating unusual choices in the Dolev-Yao models or the realizations. We also start to discuss which of the problems known for BRSIM/UC soundness extend to weaker soundness notions.

1 Dolev-Yao Models

Tools for proving security protocols typically abstract from cryptography by deterministic operations on abstract terms and simple cancellation rules. An example term is $E_{pk_{ew}}(\text{hash}(\text{sign}_{sk_{su}}(m, N_1), N_2))$, where m denotes a payload message and N_1, N_2 two nonces, i.e., representations of fresh random numbers. We wrote the keys as indices only for readability; formally they are normal operands in the term. A typical cancellation rule is $D_{sk_e}(E_{pk_e}(m)) = m$ for corresponding keys. The proof tools handle these terms symbolically, i.e., they never evaluate them to bitstrings. In other words, the tools perform abstract algebraic manipulations on trees consisting of operators and base messages, using only the cancellation rules, the message-construction rules of a particular protocol, and abstract models of networks and adversaries. Such abstractions, although different in details, are collectively called Dolev-Yao models after their first authors [15].

While Dolev-Yao models are no longer the only way of treating cryptography in automated tools, they are likely to remain important where they are applicable because of the strong simplification they offer to the tools, which enables the tools

to treat larger overall systems automatically than with more detailed models of cryptography.

2 Soundness Results

It is not obvious that a proof in a Dolev-Yao model implies security with respect to real cryptographic definitions. However, in the last years significant progress was made in showing that this is true in many cases. Early results considered passive attacks only [2, 1, 17]. The first result that allows active attacks [7] and thus the typical Dolev-Yao adversary model immediately used a very strong notion of soundness, blackbox reactive simulatability/UC [21, 22, 12]. This notion essentially means that a “real system”, here the realization, can be plugged in for an “ideal system”, here the Dolev-Yao model, safely in arbitrary environments. The result in [7] is also strong in allowing multiple cryptographic primitives. However, it makes some relatively unusual choices and one first-time addition in its Dolev-Yao model and needs additional type tagging and randomization in the realization. The result was extended to more cryptographic primitives in [8, 4] with increasing extensions to the Dolev-Yao models. General results on property preservation through the BRSIM notion imply certain other soundness notions for the same Dolev-Yao model and realization [21, 3], and additional specific soundness properties were proved in [6]. Stronger links to conventional Dolev-Yao-style type systems were provided in [19], and an integration into the Isabelle/HOL theorem prover in [23].

Later papers such as [20, 18, 13] define weaker soundness notions, such as integrity only or offline mappings between runs of the real and ideal systems, and/or allow less general protocol classes, e.g., only a specific class of key exchange protocols. For these cases, they can use simpler Dolev-Yao models and/or realizations than [7].

All the results about linking Dolev-Yao models and cryptography mentioned so far are essentially positive, i.e., soundness in some sense is shown. Furthermore, they concentrate on core cryptographic systems such as encryption and signatures; they do not contain hash or one-way functions, nor operations with algebraic properties such as XOR, although such operations exist in the Dolev-Yao models of many automated tools. Our work on impossibility was motivated by trying to add XOR and hashing to the BRSIM/UC soundness results, and being unsuccessful even if we were willing to make very significant changes or restrictions to the Dolev-Yao model, the protocol class using it, or the implementation.¹

¹ We do obtain BRSIM/UC-style soundness, e.g., for hashing in the random oracle model and for XOR in the passive case. However, we do not find these positive results fully satisfying. Results that make similar strong restrictions while not even aiming at BRSIM/UC soundness were also obtained in [10, 16].

3 Impossibility Results

Given the state of the art of Dolev-Yao soundness results, it is interesting to consider impossibility results. For instance, one may ask the following questions:

- Is it really not possible to show soundness for hashes and XOR (and probably further related primitives) in the same strong BRSIM/UC sense as for encryption and signatures?
- In cases where positive results exist both for BRSIM/UC soundness and weaker soundness, with simpler systems in the latter case, is this an unavoidable tradeoff?
- Where positive results exist only for restricted protocol classes, or simpler results than in the general case, are all the restrictions really needed to achieve these results?

While we do not claim to have the final answer to all these questions, we can present a number of results.

3.1 Hashes

In particular, we present answers to the first question for hash functions. We first show that it is indeed impossible to realize the standard Dolev-Yao model of hashing with standard cryptographic hash functions in the sense of BRSIM/UC. However, we can go significantly further. In particular, we show impossibility even if we give up the secrecy of hashed messages in the Dolev-Yao model (leaving only collision resistance – this is a reasonable possibility in Dolev-Yao models). We also show impossibility if probabilistic hashing [11, 14] is used as the realization; this cryptographic primitive offers better secrecy than deterministic hashing and can sometimes be used instead of random oracles where deterministic hashing cannot. Moreover, we discuss that many plausible restrictions of the protocol classes do not help, although for some very strong restrictions we do achieve BRSIM/UC again.² (These results will be published as [9]; the report version does not yet contain the results on probabilistic hashing.)

3.2 XOR

We also give a survey on similar results for XOR from [5]. Furthermore, we give a short introduction into how one can set up impossibility proofs that hold across the multitude of significantly different rigorous definitions of Dolev-Yao models in the literature.

² This is also a little surprising as one of these restrictions allows standard ideal Dolev-Yao style secrecy and uses standard deterministic hash functions, i.e., the “canonical” setting. However, here only individual nonces (i.e., cryptographic objects with no other purpose, in contrast to payloads, keys, ciphertexts, etc.) can be hashed. Thus essentially only one-time signatures can be produced. Then even BRSIM/UC soundness for the Dolev-Yao model does not require secrecy of the individual bits of the real nonces.

3.3 Encryption and Authentication

Furthermore, for the first time we start surveying existing informal answers to the second and third question above and to make them more rigorous. Some issues concerned are the following: the leakage of the message length through encryptions, the need to make probabilistic encryption and signatures explicit in the Dolev-Yao model through a freshness construct on the respective term type, the need to additionally randomize certain realizations because of potential problems with adversary-chosen keys, and the possibility that symmetric authentications or ciphertexts are valid with respect to several adversary keys.

We also identify gaps where no such answers exist yet, and hope to stimulate discussions and future work on those.

Acknowledgments. We thank Martín Abadi, Véronique Cortier, Anupam Datta, Ante Derek, Cathy Meadows, John Mitchell, and Andre Scedrov for interesting discussions. This work is partially supported by the European Commission through the IST Programme under Contract IST-4-026764-NOE ReSIST.

References

1. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th TACS*, pages 82–94, 2001.
2. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP TCS*, volume 1872 of *LNCS*, pages 3–22. Springer, 2000.
3. M. Backes and B. Pfitzmann. Computational probabilistic non-interference. In *Proc. 7th European Symp. on Research in Computer Security (ESORICS)*, volume 2502 of *LNCS*, pages 1–23. Springer, 2002.
4. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE CSFW*, pages 204–218, 2004.
5. M. Backes and B. Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In *Proc. 10th ESORICS*, volume 3679 of *LNCS*, pages 178–196. Springer, 2005.
6. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing*, 2(2):109–123, 2005.
7. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proc. 10th ACM CCS*, pages 220–230, 2003.
8. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th ESORICS*, volume 2808 of *LNCS*, pages 271–290. Springer, 2003.
9. M. Backes, B. Pfitzmann, and M. Waidner. Limits of the Reactive Simulatability/UC of Dolev-Yao models with hashes. In *Proc. 11th ESORICS*, LNCS. Springer, 2006. To appear. Preliminary version IACR Cryptology ePrint Archive 2006/068, <http://eprint.iacr.org/>.
10. M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proc. 32nd Intern. Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 652–663. Springer, 2005.

11. R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. In *Proc. CRYPTO 97*, pages 455–469. Springer, 1997.
12. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE FOCS*, pages 136–145, 2001.
13. R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conf. (TCC)*, pages 380–403. Springer, 2006.
14. R. Canetti, D. Micciancio, and O. Reingold. Perfectly one-way probabilistic hash functions. In *Proc. 30th ACM STOC*, pages 131–140, 1998.
15. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
16. F. D. Garcia and P. van Rossum. Sound computational interpretation of formal hashes. IACR Cryptology ePrint Archive 2006/014, Jan. 2006.
17. P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symp. on Programming (ESOP)*, pages 77–91, 2001.
18. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symp. on Security & Privacy*, pages 71–85, 2004.
19. P. Laud. Secrecy types for a simulatable cryptographic library. In *Proc. 12th ACM CCS*, pages 26–35, 2005.
20. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conf. (TCC)*, volume 2951 of *LNCS*, pages 133–151. Springer, 2004.
21. B. Pfizmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM CCS*, pages 245–254, 2000.
22. B. Pfizmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symp. on Security & Privacy*, pages 184–200, 2001.
23. C. Sprenger, M. Backes, D. Basin, B. Pfizmann, and M. Waidner. Cryptographically sound theorem proving. In *Proc. 19th IEEE CSFW*, 2006. To appear.

Model-based Automatic Test Generation for Event-Driven Embedded Systems using Model Checkers

Zoltán Micskei, István Majzik
Budapest University of Technology and Economics
Dept. of Measurement and Information Systems
Magyar Tudósok krt. 2., H-1117 Budapest, Hungary
micskeiz@mit.bme.hu, majzik@mit.bme.hu

Abstract

Testing is an essential, but time and resource consuming activity in the software development process. In the case of model-based development, among other subtasks test construction and test execution can be partially automated. Our paper describes the implementation of a test generator framework that uses an external model checker to construct test sequences. The possible configurations of the model checker are examined by measuring the efficiency of test construction in the case of different statechart models of event-driven embedded systems. The generated test cases are transformed and executed on common testing frameworks (JUnit, Rational Robot) and the effectiveness of tests are measured using code coverage metrics.

1. Introduction

Testing is a time and resource consuming activity in the software development process. Typically more than 30% of efforts should be reserved for testing activities. Generating a short, but effective test suite usually needs a lot of manual work and expert knowledge. A testing engineer's traditional tasks are writing *test cases* (input-output pairs) for the important functions, grouping them in *test sequences* and *test suites*, then executing the tests and finally analyzing the results. Test suites should satisfy various *test coverage criteria* often prescribed by standards (e.g. all states of the program have to be visited, all paths have to be executed). A criterion defines a set of *test requirements* (e. g. visiting given states, executing specific state transitions). The coverage of a test suite is the ratio of satisfied requirements and all requirements. Various specification and code-based criteria are presented in [1].

In a model-based development process test construction and test execution can be partially automated. In a (semi-)formal model all interfaces and possible input events are gathered which forms a suitable basis to start the test construction. This way model-based testing can be a solution to several problems of test generation. The following tasks can be automated:

- *Test oracle.* The model is used to derive the required output for a given test input. A quite high coverage can be achieved in a short time by randomly generating test inputs and deriving the outputs using a test oracle.
- *Estimating the coverage.* The coverage of a test suite can be estimated with the help of a system model [2].
- *Conformance testing.* Running the implementation and the model simultaneously helps to determine whether the implementation conforms to the requirements.
- *Test generation.* Complete test suites can be constructed by using the model and the test criteria defined on the specification. Typically, the state space of the model is searched for sequences of inputs (and outputs) that satisfy the test requirements. This task can be performed in several ways, e. g. implementing a specific search algorithm (like in [3]), using a Constraint Logic Programming solver, or using a model checker [4].

In our paper we present a *testing framework that uses external model checker tools to construct the test sequences*. The process used for testing is given in Figure 1. This framework is designed for *embedded event-driven systems*, where the functionality can be described as a sequence of reactions to incoming events from the environment. UML Statechart is a very popular formalism to capture the behavior of such systems. In our framework, we define test requirements on the basis of the statechart model then configure the model checker tool to find a test sequence (test inputs and required test outputs) for each requirement. The generated tests are then transformed to executable, concrete tests, and with the help of a test execution environment these tests are executed on the implementation of the model.

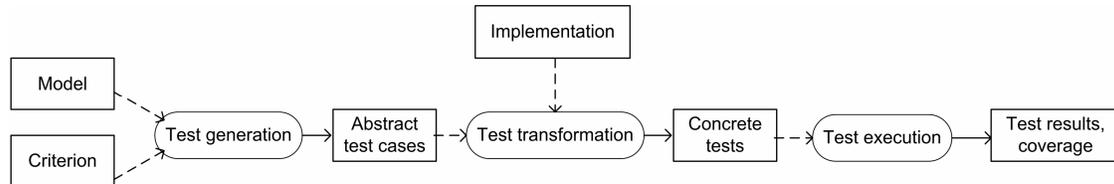


Figure 1. Testing process in our framework

There are very few publications in the literature that report on the *efficiency of using model checkers for test construction*. The configuration of the model checker in this case, namely the settings required for constructing short (and possibly minimal) test sequences, differs from the usual needs of the classical model checking problem (i.e. exhaustive verification of the full state space). In our paper we examine the possible settings of the model checker Spin by measuring the efficiency of test construction in the case of different real-life statechart models, and introduce an optimized setting for test generation. At the end of the testing process the code coverage of the automatically generated tests is measured, hence they can be compared to manually created tests.

2. Model checkers and test generation

In safety-critical systems it is necessary not only to test but also to *prove* that the system works correctly. This activity is supported by *model checker* tools. They examine the state space of the input model to check the truth of *temporal logic expressions* that apply temporal operators to express *state reachability* properties and required *temporal ordering of states*. The greatest challenge of these tools is the *state space explosion* as practical models of concurrent systems may have a huge number of states. The limit of state-of-the-art model checkers grows above 10^{20} states.

In our experiments we applied the following model checkers:

- *Spin* [5] accepts models of concurrent processes given in Promela, its specific input language that supports global variables and communication channels. The requirements can be formalized by reference automata or by LTL (Linear Temporal Logic) formulae. Partial order reduction and bit state hashing are applied to handle the state space.
- *SMV* is a symbolic model checker that accepts the description of finite state machines. (Its input language has constructs to express state transitions.) Requirements can be formalized by CTL (Computational Tree Logic, a branching time logic) formulae. Internally it uses Binary Decision Diagrams to store and manipulate the state space.
- *Uppaal* [12] is a model checker for real-time systems. Its input model can be given by a set of timed automata including global variables and conditions that refer to clock variables. Requirements are expressed in a CTL-like language.

Model checkers were proposed to be used as test generation tools in various publications. [4] used SMV to generate tests for StateMate statecharts, but as far as we know no tool was developed to automate the process. The AGEDIS project aims at the automation of testing. The developed tool generates tests from specially annotated UML diagrams, and the abstract test suites can then be mapped to concrete ones executed on the implementation [6]. In [7] mutation analysis was used to create tests that can detect different programming errors. The method presented in [8] is similar to [4], but here Spin is used and the input model is an abstract state machine. A prototype of a test generator tool (ATGT) is also reported. There are several other academic and commercial tools available as summarized in [9].

In our experiments we apply the following test generation method:

1. The system model in the form of annotated UML statechart is transformed into the input language of the model checker tool.
2. Each test requirement of a given coverage criterion (e.g. reachability of a given state by a test sequence) is formulated as a temporal logic expression.
3. For each expression the negation of the formula is verified by the model checker. If there is an execution path in the model that does not satisfy the negated formula (e.g. it turns out that a given state can be reached by an execution path) then it is presented by the model checker as a counter-example. This path becomes a test sequence that satisfies the original test requirement.
4. The inputs and outputs that form the executable test sequence are extracted from the counter-example or derived by a corresponding guided simulation of the model.

The necessary model transformation and the configuration of the model checker are detailed in the next sections. The temporal logic based formulation of the set of test requirements is illustrated in the case of the following coverage criteria:

- *All states* coverage: For each state of the statechart a test sequence is generated, that leads the system to that state. The LTL formula-set for this criterion is

$$\{ \neg(F \text{ in}(s)) \mid \forall s \in S \} \quad (1)$$

where F is the eventually operator in LTL, S is the set of states and $\text{in}(s)$ is a Boolean expression on the state variables of the model that is true when the state s is active.

- *All transitions* coverage: For each transition in the statechart a test sequence is generated that fires the transition. The LTL formula-set for this criterion is

$$\{ \neg(F \text{ firing}(t)) \mid \forall t \in T \} \quad (2)$$

where T is the set of transitions in the model and $\text{firing}(t)$ is a Boolean expression on transition variables that is true when transition t is fired.

Other control-oriented test requirements (e.g. coverage of all configurations, transition-pair coverage) can be formulated similarly. Data-oriented criteria (e. g. all definition-use paths) are not covered by our experiments. The temporal logic representation of the corresponding test requirements can be found in [10] and in [4].

3. The test generation framework

Our framework consists of the tools and data files presented in Fig. 2. The box marked with TR is a model transformation from UML statecharts to Promela [11] that uses *Extended Hierarchical Automata* (EHA) as its intermediate format. The transformation consists of two programs: *sc2eha* (in Prolog), and *eha2promela* (in Java). The box marked with TG is the test generator implemented in Java. Its input parameters are the statechart model and the test coverage criteria. It controls the test generation process as follows:

1. The settings of the tools are read from an XML based configuration file.
2. Both `sc2eha` and `eha2promela` programs are executed. Here `sc2eha` builds an EHA representation of the model which is accessed by the test generator to obtain information on the events, states and transitions in the statechart.
3. For each test requirement a file containing an LTL formula is created.
4. The executable `pan` of the model checker Spin is executed that produces a report of the results and a trail file (i.e. a counter-example used later as the test sequence), if exists.
5. A filtering procedure generates an XML file that contains only the input and output operations from the sequence of atomic actions described in the trail.
6. The temporary files are preserved, in this way if no test is found or Spin runs out of memory then the test generation can be repeated with modified settings.

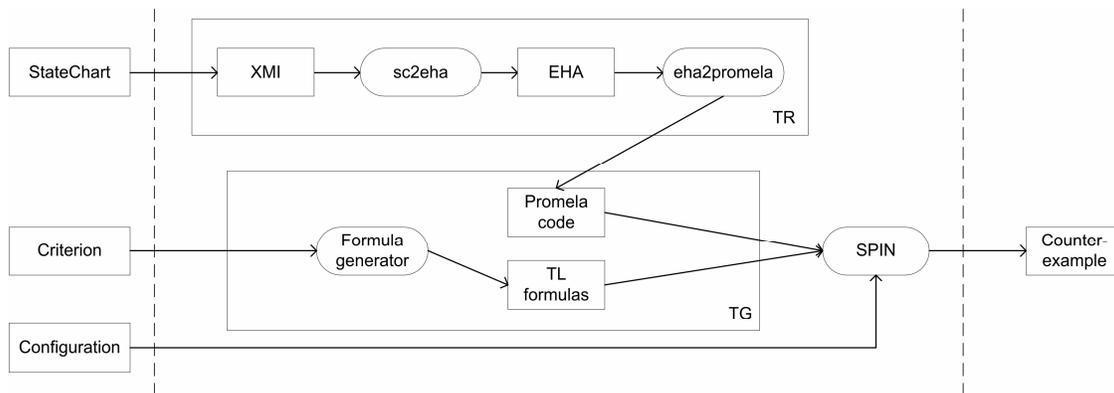


Figure 2. Test generator framework

4. Configuration of the model checker

Classical model checking aims at the fast *exhaustive search of the full state space* of the model. In contrary, test generation by model checking aims at the fast and efficient *construction of a counter-example* by visiting and storing *as few states as possible*. Not all counter-examples are needed, however the one that is generated should be minimal in length. Due to this difference, the direct (default) use of model checkers typically results in poor efficiency when used for test generation. Fortunately, the well established tools offer several built-in state handling techniques and parameters for tuning that make the optimization of the test construction possible. We performed several experiments to determine the effects of the available options.

The experiments were performed on a 2 GHz PC with 512 MB RAM. The benchmark model was a statechart describing the operation of a mobile phone having 10 states, an event queue and 21 transitions (about 500000 state configurations are visited by Spin).

Table 1. Execution time of test construction phases

	Generate LTL formula	Generate pan	Compile pan	Run pan	Extract tests
Default case	0,25s	0,70s	44,92s	75,00s	0,71s
Short tests	0,23s	0,69s	44,37s	553,51s	0,66s

Table 1 shows the execution time of two typical test generation runs. The first case is the default configuration with no test-specific settings, while in the second case Spin was

instructed to find iteratively the shortest counter-example. This is why the execution of the model checker executable *pan* lasted longer (while the time required for formula generation and compilation remained roughly the same). It is important to note that in shorter runs the compilation time and the running time are in the same order of magnitude.

Both the compilation and the execution of *pan* are controlled by several options [5]. The settings interesting from the point of view of test generation are as follows:

- Breadth first search: The compilation option `-dBFS` means that instead of the default depth first search (DFS) algorithm *breadth first search* (BFS) is implemented. It is optimal for test generation since a breadth first search finds the shortest trail. However, the depth of BFS is limited by the available memory. Using the default settings only a three transition long sequence could be generated. BFS is capable of generating all test sequences if the lengths of the event queues are reduced in the model and atomic instruction sequences are used.
- Limited DFS: The run-time option `-m` sets a *limit for the DFS*. It is one of the most useful options, since it restricts the depth of the search (i.e. the length of the counter-example). However, if it is set too low then test sequences cannot be found. In this case it is helpful to increment the value until the memory limit is reached.
- Iterative search for short traces: `-I` and `-i` options *search for shorter counter-examples* with iterative runs. They could greatly enlarge the execution time as `-i` finds the shortest trail. `-I` is approximate and faster, it performed well in our experiments.
- Model dependent options: The usage of `-dNOFAIR` (weak fairness), `-dSAFETY` (cycle detection disabled) parameters did not result in significant improvements.

We found that the following settings are optimal for this model (see in Table 2):

- DFS search: All compiling options are used except `-dBFS` and `-dREACH`.
- No iterative search: Options `-I` and `-i` are not used since they increase the time required for test construction. Instead of them, DFS is limited by parameter `-m`.
- Limited DFS: Option `-m` is set to the minimal value where tests are generated (it can be set in a few probe runs). Changing the value from 500 to 200 resulted in 80% decrease of the execution time in the benchmark model.
- Hash table size: Option `-w` controls the size of the hash table used to store the visited states. The value of this switch should be increased in the case of detecting a high number of hash conflicts. In this example the value 24 resulted in a total memory usage of 67 MB.

Table 2. Execution time and test properties in case of different options.

Options	Time required for test generation	Length of the test sequences	Longest test sequence
<code>-i</code>	22m 32.46s	17	3
<code>-dBFS</code>	11m 48.83s	17	3
<code>-i -m1000</code>	4m 47.23s	17	3
<code>-I</code>	2m 48.78s	25	6
Default	2m 04.86s	385	94
<code>-I -m1000</code>	1m 46.64s	22	4
<code>-m1000</code>	1m 25.48s	97	16
<code>-m200 -w24</code>	46.7s	17	3

Using these parameters the generation of test sequences covering all states required 46.7 seconds, while covering all transitions required 4 minutes and 19 seconds. All test sequences were the shortest possible (this was cross-checked manually).

The lessons learnt during these experiments can be summarized as follows:

- The original (default) settings resulted in relatively short execution time but overlong test sequences (the length is 385 instead of 17).

- Iterative search algorithms resulted in short test sequences but long execution time (22 minutes vs. 2 minutes). The same applied to BFS (11 minutes vs. 2 minutes).
- The best results were obtained in the case of limiting the depth of the DFS together with setting a limit to the hash table size.

It turned out that the suitable setting of options could reduce the time required to find the necessary test suite to 37% (comparing to the default case) and to 3.5% (comparing to the pure iterative search). The obvious settings (iterative search) did not perform well.

Additional experiments were performed to compare the efficiency of Spin to SMV. The total execution time of generating test sequences covering all states was 10.98 seconds. SMV outperformed Spin due to the following reasons: In SMV there is no need to compile the verifier and SMV reuses the data structures between different verification runs. However, the Spin model, thanks to the automatic generation, is more generally applicable.

5. A case study

The applicability of the framework was demonstrated in the case of a real-life industrial model. We choose a protocol that synchronizes status and control bits between two computers in a distributed control system in presence of anticipated faults. The model consists of 5 objects with event queues, 31 states and 174 transitions. In the generated Promela code the state vector (which identifies a state) was 216 bytes long, and during the exhaustive verification more than $2 \cdot 10^8$ states were explored. The complete verification would need approximately 40 GB of memory, therefore the application of state compression techniques was necessary.

We examined the effects of the *bit-state hashing* state compression technique offered by Spin. This technique does not store the entire state vector, only one or two bits per state in the memory. This results in states that are not stored separately but merged with others. Additionally to the state compression technique, the lengths of the event queues of the objects were decreased to the minimum. This kind of reduction modifies the behavior of the model by reducing the length of the sequences that can be produced.

The reduction and compression techniques are applied in a *conservative manner*. On the one hand, if a test sequence is found in a reduced model (note that typically there are several tests for a given requirement) then it is a valid test sequence executable on the full model. On the other hand, if no test sequence is found for the given requirement then it may happen that a test sequence can be generated on the basis of the full model.

Applying these reductions the test generation was successful, but it required two and a half hours to terminate. Two more adjustments were needed:

- Several test requirements are satisfied by multiple test sequences. Accordingly, the number of the verification runs can be reduced if each test sequence is checked for covering other test requirements.
- Parameter *-w* (size of the hash table) is more important in bit-state mode. If it is too low, the verification misses too many states, and thus test sequences are lost.

Table 3. Test generation for the synchronization protocol considering coverage of all states.

Parameters	Execution time	Redundant runs skipped	No test found (states)
-m1000 -w31	65m4.32s	65 %	0
-m1000 -w26	46m2.30s	62 %	2
-w26	56m47.66s	55 %	4
-w24	28m3.35s	48 %	8

Table 3 shows that our tool (with proper parameters) can be used to construct test sequences for complex protocols with huge state space. In this case the test suite consists of 31 test sequences if a separate test is generated for each state. The length of a test sequence is typically 11 steps. If the redundant tests are eliminated then 3 long and several short sequences (typically 3-4 steps) are required.

6. Applying tests to implementation

The tests generated by our framework are *abstract*, i.e. they correspond to the events and actions defined in the model. If we want to use them to test concrete implementations, then the test cases should be mapped to *concrete* test cases which are executable on the real System Under Test (SUT). The process of this transformation was demonstrated on the mobile example, constructing a Java GUI based implementation and a console-based implementation (this latter was generated automatically). A concrete test case contains the following parts:

- Setup code to establish the necessary state required to start the test.
- Code for sending the events, and checking that expected actions can be observed.
- Teardown code to clean up any resources created by the test.

Our transformation used *templates* to automate concrete test case generation. One template contained the general structure of the test case file, another included the parameterized code used to send events and check actions. Test cases were generated for the *JUnit* Java unit testing framework and for the *Rational Robot* automatic GUI testing tool. Only 10 lines of implementation-specific code had to be inserted into the templates to generate approximately 500 lines of test code.

One of the most important metric to evaluate the efficiency of a test suite is the code coverage it produces. We measured the statement, method and condition coverage the test suite produced for both programs. Figure 3 shows the results.

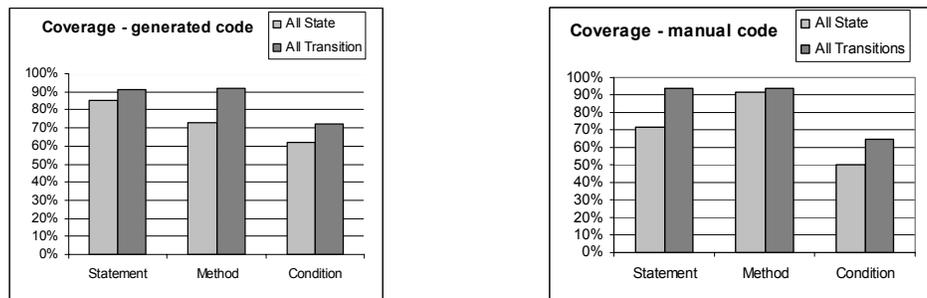


Figure 3. Code coverage results

The abstract test suite (that was characterized by the “all transitions coverage” criterion in the model) covered in the code more than 90% of the statements and methods. The remaining 10% was exception handling code (e.g. handling invalid events), which was not called because test cases contained only valid events. These experiments show that our approach is a cheap and efficient way to generate a base test suite quickly that exercises the majority of the implementation’s code.

7. Testing Real-Time Systems

In the case of real-time systems test sequences shall carry timing information, i.e. the *time delay between successive test inputs* (and timing parameters of the required outputs). The

execution of such a test sequence shall keep these time delays in order to execute the given path determined by the coverage criteria.

In our framework test sequences can be generated on the basis of *UML statechart models extended with timing information* (clock variables). In this case the model checker Uppaal [12] is applied that is capable of verifying real-time systems.

Uppaal offers the following functions which are useful for test generation:

- BFS can be used (besides DFS). It provided the shortest tests for relatively small systems.
- It has multi-level state compression and supports bit-state hashing as well.
- Uppaal can reuse existing data structures when multiple properties are checked.
- It can generate the shortest trail without iterative runs (disables reuse).

As a first benchmark, the behavioral model of the mobile phone example was extended with timing information. The transformation from timed UML statecharts to the timed automata of Uppaal faced the following challenges:

- There is no state hierarchy in Uppaal. The transformation should ensure that a sub-automaton is active only when its parent automaton is active.
- There are only binary synchronization channels in Uppaal, which cannot pass values. Event queues were implemented using global arrays and dedicated dispatcher processes.

The results of the first experiments with the mobile phone example were encouraging. The test generation took 24.94 seconds and Uppaal generated the shortest possible test sequences. Uppaal recorded in the trail (counter-example) whenever a delay occurred between the firings of transitions in the model, so the timing information (i.e. the delay between successive test inputs) required executing the test sequence could be extracted from the trail file.

8. Conclusion

We outlined a framework that uses external model checkers to construct test sequences corresponding to state and transition coverage criteria in event-driven and real-time systems. Different configurations of the Spin model checker were examined experimentally to find those settings that are optimal for test generation. We demonstrated that (i) using proper options the time required for test generation can be significantly reduced, (ii) the state compression techniques do not hinder the construction of test sequences in practical models of concurrent systems, and (iii) the generated abstract test produces high coverage on the implementation.

9. References

- [1] R. Binder: *Testing Object-Oriented Systems*, Addison-Wesley, 1999.
- [2] P. Ammann et al: "Model Checkers in Software Testing", NIST-IR 6777, NIST, 2002.
- [3] W. Grieskamp et al: "Test Case Generation from AsmL Specifications. Tool Overview". 10th International Workshop on Abstract State Machines, March 3-7, Italy, 2003.
- [4] H. Seok Hong, I. Lee, O. Sokolsky, S. Deok Cha: "Automatic Test Generation from Statecharts Using Model Checking", MS-CIS-01-07, University of Pennsylvania, Feb 2001.
- [5] G. Holzmann: *The Spin Model Checker Primer and Reference Manual*, Addison-Wesley, 2003
- [6] A. Hartman et al: "The AGEDIS Tools for Model Based Testing", ISSTA 2004, 2004.
- [7] P. Ammann: "Using Model Checking to Generate Tests from Specifications", ICFEM, 1998.
- [8] A. Gargantini: "Using Spin to Generate Tests from ASM Specifications", ASM03, 2003.
- [9] A. Hartman: "Model-based test generation tools", <http://www.agedis.de>, 2003.
- [10] H. Seok Hong et al: "Data flow testing as model checking", In Proc. ICSE, 2003.
- [11] D. Latella et al: "Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-checker". *Formal Aspects of Computing*, (11)6, pp 637-664, 1999.
- [12] T. Amnell et. al.: „Uppaal - Now, Next, and Future". In Proc. *Modelling and Verification of Parallel Processes (MOVEP'2k)*, LNCS 2067, pages 100-125, Springer Verlag, 2000.

ROBUSTNESS TESTING TECHNIQUES FOR HIGH AVAILABILITY MIDDLEWARE SOLUTIONS

ZOLTÁN MICSKEI, ISTVÁN MAJZIK

*Department of Measurement and Information Systems
Budapest University of Technology and Economics, Budapest, Hungary
{micskeiz, majzik}@mit.bme.hu*

FRANCIS TAM

*Nokia Research Center, Nokia Group, Finland
francis.tam@nokia.com*

To increase the interoperability of availability management software (also known as high availability middleware) the Service Availability Forum has released a set of open specifications. With the development of a common interface the comparison of multiple products can be achieved. For high availability (HA) solutions, assessing the robustness of the HA middleware is as important as measuring its performance. This paper investigates the sources of inputs that can activate robustness faults of a HA middleware and recommends the corresponding testing techniques to check the existence of such faults. We investigated the automated construction of the robustness test suites and compared the efficiency of different techniques using a case study with an open-source HA middleware.

1. Introduction

In the past few years dependability became a key attribute even in common computing platforms. High availability (HA) can be achieved by introducing *redundancy* in the system, like warm standby spares, redundant communication channels etc. The configuration of the redundant components, thus the management of the availability of the whole system, is often application independent. The necessary services (e.g. membership, recovery) can be implemented as a generic middleware.

To increase the interoperability of availability management software (known as HA middleware) major users and vendors formed a consortium, the Service Availability Forum with the goal to develop open specifications for availability management of software and the underlying hardware. SA Forum's Application Interface Specification (AIS) [1] defines the interface between the HA middleware and the custom application. It is a C language interface partitioned

into a number of services. For example, the Cluster Membership Service (CLM) provides a consistent view of the computing nodes, while the Availability Management Framework (AMF) manages the availability of redundant components. Three major versions have been released for AIS so far, the latest being B.02.01. There are several implementations available for AIS; we used in our experiments an open-source middleware, OpenAIS [2] (alpha release, version 0.69).

Having a common specification for the HA middleware products, the demand to compare the various implementations naturally arises. Most of the comparisons and benchmarks of similar middleware products address performance, but in case of a HA middleware, the *robustness of the implementation* is also a crucial attribute. Robustness failures in the middleware can be activated by poor quality application components, and one such component may render the whole application inaccessible. Thus, our long-term goal is to define a method to evaluate and compare the robustness of different AIS based HA middleware implementations.

Robustness is a secondary attribute of dependability and it is used in this paper as defined in [3], i.e., the degree to which a system operates correctly in the presence of *exceptional inputs* or *stressful environmental conditions*. Accordingly, *robustness faults* are those faults that can be activated by these inputs and conditions, resulting in an incorrect operation (e.g. crash, deadlock) of the system.

Although there is an open-source implementation of AIS, most of the implementations are (and will be) commercial products with limited information about their internal structure. Without a detailed behavioral model or source code, the evaluation can only be based on the common interface specification. Accordingly, the services (functions) defined by the AIS can be tested for robustness faults externally by executing specific test sequences called *robustness tests*. The approach of robustness testing is similar to functional “black box” testing, but it concentrates on the activation of potential robustness faults by providing exceptional inputs and generating stressful conditions. Thus the basis of the comparison of AIS implementations is the common interface specification, as the number of robustness faults per functions is measured.

Robustness testing is a time and resource consuming activity. Generating an effective test suite, executing it and evaluating the results usually needs a lot of manual work. In a model-based development process test construction and test execution can be partially automated. AIS provides a semi-formal description of the interfaces, which can be used to gather the possible inputs and output acceptance conditions, and thus it allows automated test construction and test

execution. Moreover, this interface specification can be utilized to construct more sophisticated test sequences than the commonly used ones (based on input variable domains only). Accordingly, in this paper we focus on the following aspects of robustness testing:

- *Automated construction of robustness test suites for AIS based HA middleware.* The exceptional input values are generated by automated tools on the basis of the functional specification.
- *Elaboration of extended robustness testing techniques.* Scenario-based robustness testing techniques are proposed to cover non-trivial robustness faults in state-based functions of the AIS. In the case of these functions a specific call sequence is required to reach the state in which the service can be used, otherwise a trivial error code is returned without executing the service and thus activating the potential robustness faults.
- *Evaluation of the test results using intelligent data processing techniques.* On-line analytical processing and basic data mining methods are proposed to identify the key factors (e.g. product version, OS version, workload) that influence robustness.

In the paper Section 2 summarizes the previous robustness testing projects. In Section 3 the concepts of our robustness testing framework for AIS-based HA middleware are presented. The different robustness testing techniques are described in Section 4 and 5. The efficiency of the techniques is compared in Section 6. Finally, Section 7 concludes our results and lists future plans.

2. Related work

Robustness testing was the goal of several research projects in the past. Different methods were applied to measure the dependability of complex systems at various abstraction levels.

Fuzz [4] was one of the first tools designed especially for robustness testing. It utilized randomly generated character strings to test common UNIX console utilities. This simple method found for 20% of the tested 80 applications an input sequence that crashed the program.

The Riddle tool [5] was used to test the operating system API in Windows NT. Two techniques were applied for input generation. The *generic technique* used a fixed input domain for all parameters of the API while the so called *intelligent one* used a specific generator for each type. The tests found abort failures in 10% to 80% of the functions in three system DLLs. The four-year Ballista project [6] assessed the robustness of POSIX API implementations and conducted a great number of experiments on 14 UNIX versions. The robustness

test suite, which also applied type-specific input generators, was used mainly for comparing the different UNIX products. Later the method was extended for CORBA, Windows and for a simulation backplane testing.

The goal of the recent *dependability benchmarking projects* was slightly different; they defined benchmarks to characterize the system behavior under typical load and common fault conditions. A general framework for creating dependability benchmarks was developed in the EU project DBench [7]. The method was implemented e.g. for operating systems [8]. Software and hardware vendors are also providing availability benchmarks for their products, e.g. IBM for autonomic computing [9] and Sun for the R-cubed framework [10].

In our work we tried to integrate the complementary solutions for robustness testing and extend them with advanced methods specific to HA middleware.

3. The AIS robustness testing framework

The first step of defining the testing strategy in the case of a “black box” AIS middleware is to identify the possible sources of inputs that can activate robustness faults. These inputs are depicted in Figure 1(a). In the following the potential robustness faults are grouped on the basis of the source of activation, defining in this way the *type* of the fault. For each fault type a testing technique was selected as shown on Figure 1(b):

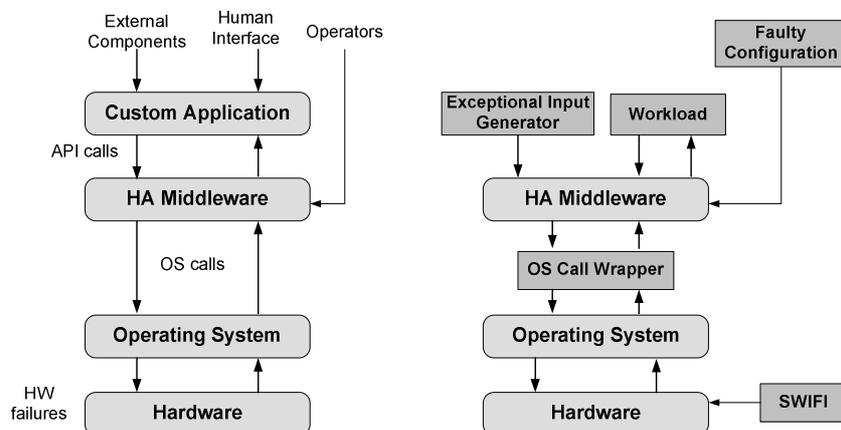


Figure 1. Inputs that can activate robustness faults in a HA middleware (a) and the proposed robustness testing techniques (b)

- The calls from the custom application (which propagate the effects of human operators and external components as well) are provided by an

exceptional input generator and a background *workload*. The workload represents the typical operation of the system. Note that in case of a HA middleware it should include failovers, administrative restarts and other fault-masking activities, because they are part of the normal operation.

- Configuration inputs (given by system operators) are represented by providing *faulty configurations*.
- Exceptional results of operating system (OS) calls are given by an *OS call wrapper* that catches the return values of OS calls, injects the exceptional return values defined by the *faultload* and provides observability. The *faultload* defines the type and timing of the injected faults. Note that simulating failures of operating system calls has two purposes. It checks the reaction of the HA middleware not only in case of a fault in the OS itself (which has quite low probability), but also in the case of many other failures in the environment (e.g. wrong file access settings, insufficient resources) that are manifested in exceptional results from OS calls.
- Hardware level *faultload* is provided by *software implemented fault injection* (SWIFI).

These techniques can be executed in two distinct phases of testing: (1) testing the API with a robustness test suite containing exceptional inputs, (2) workload based benchmarking with injected faults representing stressful environmental conditions.

In our paper we focus on the first phase of testing. The selected techniques are supported by the following set of tools that allow an automated construction of test suites containing exceptional inputs:

- *Template-based type-specific test generator* tool. Templates specify type and function information on the basis of the AIS API and the tool generates the test programs automatically (see in Section 4).
- *Scenario-based sequential test generator* tool. To construct test sequences needed to test state-based API functions, besides the AIS specification the *functional test sequences* provided by the vendors of the HA middleware were also utilized. The tool will process these sequences, and executes (1) parts of them to reach specific states in which type-specific test inputs can be used and also (2) applies mutation operators (e.g. changing the sequence of tests, modifying parameters or function names) to construct exceptional sequences (see in Section 5).

In the second phase of testing the stressful environmental conditions can be provided by implementing a *workload* with a *faultload* (as in other previous dependability benchmarks). The following tools are proposed:

- *Faulty configuration generator* tool. The administrative interface of AMF was introduced recently (January 2006), but the configuration was not standardized, in this way this tool could not be realized. As soon as products will support this specification, the (mutation-based) administrative actions and faulty configurations can be generated and executed.
- *OS call wrapper*. OS level errors are injected by a wrapper between the OS and the middleware, like in [11]. A lightweight wrapper can be implemented on Linux with the LD_PRELOAD environmental variable, which can be used to reroute the system calls to modified libraries.
- *SWIFI tool*. Besides explicit component failures, like abrupt node shutdown or network interface failure, lower level hardware faults can be injected by external tools like FAUmachine (formerly UMLinux [12]).

One of the most labor-intensive part of robustness testing is the evaluation of the test outcome. Functional test cases usually contain the expected result and compare the actual result to this reference value. In case of robustness testing there is a widely accepted *simplified approach*. Obvious robustness failures, i.e. crash and abort-like answers are recognized. However, no differentiation is made between the other possible results, i.e., successful answer, valid error code according to the specification, misleading error code and silent errors. (This simplification was necessary in most systems to reduce testing costs and avoid the problems originating in missing or incomplete specification, especially in case of erroneous behavior.) We refined this method by assigning the possible error codes (as potential results) to test inputs values. The test outputs are then filtered and only those test runs are inspected, in which the output was not among the expected error codes.

Even in our early tests thousands of robustness test cases were generated, thus an automated method was needed to analyze the results. Two previously recommended techniques were used to accomplish this. Online Analytical Processing (OLAP) was applied to filter and compare the results of different systems [13] and data mining to identify the possible fault sources [14].

In the following sections we describe the implemented tools and techniques of API testing.

4. Generic and type-specific testing

Exceptional inputs can be grouped into the following categories:

- *Syntactically not correct values*: e.g. invalid string for an IPv4 address.
- *Semantically not correct values*: e.g. non-existing version number.
- *Values used in invalid context*: e.g. not initialized handle.

The majority of types used in API functions is defined by complex structures. Constructing exceptional values from all possible combinations of the *basic types* in these structures, like int, char, etc., would result in far too many values, because many structures are built from more than four basic types and the AIS functions have on average two or three parameters. Thus, finding good exceptional values is not as obvious as for example in case of the basic types like integers. The following subsections describe the two techniques that were used for generation of exceptional inputs for individual API calls.

4.1. Generic input testing

In the case of *generic input testing* the same set of values are used for all parameters of basic types. In C language, most of the basic types can be represented and cast to a four-byte number, as Listing (1) illustrates.

```
int paramValues[] = {0, -1, (int) &validAddress};  
...  
SaAmfHandleT * param1 = (SaAmfHandleT *) paramValues[i];  
SaNameT * param2 = (SaNameT *) paramValues[j];
```

 (1)

A few values can result in a huge number of test cases in complex structures, however, if the values are not chosen carefully, the resulting failures would not be related to robustness. The efficiency of the following values was examined.

- *0*: It is a common test value since it represents a NULL when cast to a pointer. Using zero as an input caused many segmentation faults in OpenAIS 0.69 because in the A.01.01 version of the specification many parameters are pointers and in several functions the checking of the NULL value was not implemented yet.
- *-1, 1*: These values are helpful when there are parameters of integer (or float) type. However, in the case of pointers they will be cast to memory address usually reserved for the system. When de-referenced, they cause a segmentation fault, which is surely a robustness failure, but, as far as we know, this kind of invalid pointers cannot be checked in the API functions without specific compiler extensions.
- *Random value*: Random values are popular in robustness testing, however, the repeatability of the tests is not guaranteed.
- *Address of a valid variable*: We added this value for the sake of combining exceptional values with valid values (in case of pointers of variables). In this way the sensitivity to exceptional values of function parameters can be checked one by one.

Finally, we used two input sets. The first set {0, -1, 1, fixed random} resulted in several robustness failures but in this case the failures could not be traced back to the individual parameter values (i.e., which one activated the failure) since all values used in the function calls were (potentially) exceptional ones. The second set {0, valid address} was used specifically to determine which functions failed to implement null value checks.

4.2. Type-specific testing

In the case of *type-specific testing*, unique test values were constructed for each type used in the API. The following techniques were used to enhance the efficiency of this method.

Establishing type hierarchy: The types inherit the test values of their ancestors. This technique was very effective in Ballista. In AIS there are only a few types having ancestors in the type hierarchy, so this technique was used mainly for defining a basic type with common exceptional values.

Chaining of methods: This technique was introduced in JCrasher [15]. A call graph of methods is built, where an arc between two methods represents that an output of a method can be used as an input for the other. We applied this method on the AIS AMF in case of two functions (SaAmfInitialize and SaAmfCompNameGet) that produced output for others.

Identifying valid test outputs: We observed that for some test values valid test outputs can be a priori identified. E.g. using the exceptional value 'D.5.4' for SaVersionT could result in SA_ERR_VERSION. Similarly, the results of obvious exceptional values like e.g. not initialized handle, not valid version, non existent component name, can be identified and this information can be used to classify the test results reducing in this way the number of undecided tests.

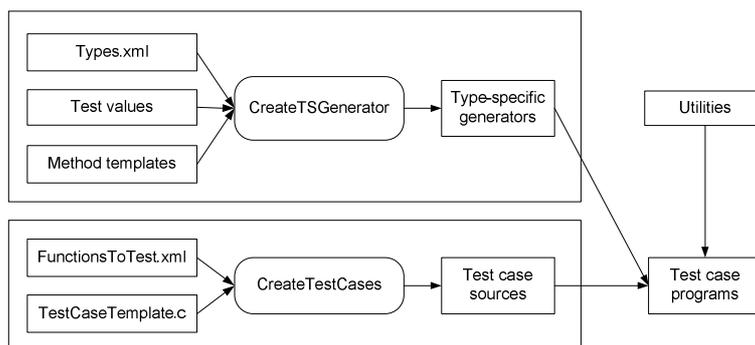


Figure 2. Architecture of the testing framework

Template-based test generation: The generic and type specific tests are implemented as separate C programs for each API function. Each program calls the API function with all combinations of the values returned by the *input value generators* and forks a new child process for each test case. The architecture of the testing framework is detailed in Figure 2. The type-specific input generators and the test sources are constructed automatically, based on templates as follows.

CreateTSGenerator constructs the C code for the type-specific input value generators. It uses the following sources and parameters:

- The *metadata of the types* for which exceptional values should be generated (types.xml, an example is found in Listing 2). Here ValidValueMethod designates the index of a valid test case. PointerMethod can initiate the construction of a method to access test values via pointers. If ParentName is present, all test cases of the given ancestor type are re-used.
- The *exceptional test values* stored in stand-alone files as C code snippets.
- The *C skeleton of the generator* and the templates for the methods.

```
<Type>
  <Name>SaDispatchFlagsT</Name>
  <ValidValueMethod generate="true" validValueIndex="1"/> (2)
  <PointerMethod generate="false" />
  <ParentName value="BaseType"/>
</Type>
```

The test case sources are constructed by CreateTestCases which is an XSL transformation that uses the following input files:

- *Test case templates* to be populated with test values.
- Information about the *API functions and their parameters* (FunctionsToTest.xml, an example is given in Listing 3). ParameterOrder is included explicitly, and IsPointer identifies whether the parameter is a pointer or not. In this way the later transformation will be easier.

```
<Function name="saAmfFinalize">
  <ReturnType>SaAisErrorT</ReturnType>
  <Parameters>
    <Parameter>
      <ParameterOrder>1</ParameterOrder>
      <ParameterName>amfHandle</ParameterName>
      <ParameterType>SaAmfHandleT</ParameterType> (3)
      <IsPointer>true</IsPointer>
      <Type>in</Type>
    </Parameter>
  </Parameters>
</Function>
```

Finally, the input generators and the test case sources are compiled and linked with a utility library, which contains functions for logging the results.

5. Scenario-based testing

The previous techniques tested individual API calls without considering that the service of several AIS functions depends heavily on the current state of the middleware and they can only be used when a sequence of previous calls have set a specific state. These call scenarios could be obtained from two sources.

- The AIS specification contains several *sequence diagrams* that capture the basic operation of the system. Using a *model-based approach*, these diagrams are re-drawn as UML sequence diagrams and the skeleton of the call sequence is generated automatically.
- The other source is the *functional test suites* of the AIS implementations. There is a public test suite, SAF Test [16], which is an open-source project for testing the conformance to SA Forum's specifications. It includes the call sequences as C test programs that can be re-used for our purposes.

When a set of scenarios is constructed from the above sources, it could be used for two purposes. First, it can be used to *reach specific states* needed by the API functions. The scenario containing the function to be tested is selected and the execution sequence preceding the call of this function is applied before initiating the generic or type-specific tests. Second, additional test cases can be generated with the help of *mutation operators* that may activate robustness failures: substituting a pointer parameter with NULL, removing a call from the scenario and changing the order of function calls.

6. Efficiency of the testing techniques

The goal of our first experiments was to compare the *effectiveness of the techniques* and to highlight the advantages of implementing the testing tools.

The tests were executed on the AMF (17 functions) and CLM module (7 functions) of OpenAIS 0.69. Table 1 illustrates the complexity (and cost) of the generic and type-specific testing techniques. The initial version of the generic testing was created in approx. three days, while the implementation of the framework of type-specific testing required about two weeks. The main advantage of the automated testing approach is that the type-specific testing of a new function requires only the completion of the metadata, and supplying the test values and logging code for the new types used in the function.

Table 1. Number of lines in the source code of the robustness testing framework

Technique	Test template	Transformations	Metadata	Test values	Sum
Generic	120	80	417	1	618
Type-specific	323	690	726	254	1993

Table 2 lists the ratio of API calls that resulted in robustness failures and the number of test calls executed. (In case of functions with more than five complex parameters the number of test cases was limited to 4000.) CLM was more resilient to generic testing since it used less pointers than AMF.

Table 2. Comparison of the different exceptional input generation and testing techniques

Technique	OpenAIS AMF	OpenAIS CLM
Generic testing with invalid addresses	2406 / 2456	60 / 424
Generic testing with null and valid address	87 / 136	0 / 44
Type specific testing	8001 / 13640	65 / 2280

In case of several functions type-specific testing identified additional robustness faults in comparison with generic testing, while in case of three functions only type-specific testing was effective (Table 3). Scenario-based testing was necessary e.g. in case of initializing callback functions.

In our experiments the *decision tree method* of a data mining tool (IBM Intelligent Miner) was used to trace back robustness failures to faults, hence a metric to compare OpenAIS with different implementations in the future was obtained. In this way, the influencing factors could also be separated.

Table 3. Faults found in OpenAIS by functions. X + Y means that generic testing found X faults while type-specific identified Y more. The star denotes a critical error, which caused segmentation fault in the middleware executive.

Function name	Faults	Function name	Faults
saAmfCompNameGet	1	saAmfProtectionGroupTrackStop	2
saAmfComponent	1	saAmfReadinessStateGet	1 + 1
CapabilityModelGet		saAmfResponse	1*
saAmfComponentRegister	2	saAmfSelectionObjectGet	1 + 1
saAmfComponentUnregisterRegister	2	saAmfStoppingComplete	1*
saAmfDispatch	1	saCImClusterNodeGet	0 + 1
saAmfErrorCancelAll	1	saCImClusterTrack	0 + 1
saAmfErrorReport	3	saCImClusterTrackStop	0
saAmfFinalize	1	saCImDispatch	0
saAmfHStateGet	2	CImFinalize	0
saAmfInitialize	0 + 2	saCImInitialize	0
saAmfPendingOperationGet	1	saCImSelectionObjectGet	0
saAmfProtectionGroupTrackStart	2		

7. Conclusion and future work

Our paper discussed the problem of robustness testing of high availability middleware. We proposed a testing framework that integrates previous testing techniques and extends them by introducing tool-supported methods including scenario-based testing and test result classification. The case study conducted on

OpenAIS showed that while even simple techniques can identify robustness problems, it is necessary to implement the more complex methods, since they are able to find faults not detected by the simple techniques. In the future we plan to apply stressful environmental conditions and we will run the test suite on other AIS implementations to compare the robustness of the different products.

References

1. Application Interface Specification, Service Availability Forum, Feb. 2006., URL: <http://www.saforum.org/>
2. OpenAIS, AIS implementation, URL: <http://developer.osdl.org/dev/openais/>
3. *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Standard 610.12.1990, 1990, URL: <http://standards.ieee.org/>
4. B. Miller *et al.*, “Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services,” Tech. Report, University of Wisconsin, 1995.
5. M. Schmid, F. Hill: “Data Generation Techniques for Automated Software Robustness Testing”, in *Proceedings of the International Conference on Testing Computer Software*, Washington, USA, June 14-18 1999.
6. P. Koopman *et al.*, “Automated Robustness Testing of Off-the-Shelf Software Components,” in *Proc. of Fault Tolerant Computing Symposium*, pp. 230-239, Munich, Germany, June 23-25, 1998.
7. H. Madeira *et al.*, *DBench Dependability Benchmarks*, project full final report, IST-2000-25425, 2003, URL: <http://www.laas.fr/dbench/>
8. K. Kanoun *et al.*, “Benchmarking Operating System Dependability: Windows 2000 as a Case Study,” in *Proc. of 10th Pacific Rim Int. Symposium on Dependable Computing*, Papeete, French Polynesia, 2004.
9. IBM Autonomic Computing, 2006., URL: <http://www.ibm.com/autonomic/>
10. J. Zhu *et al.*, “R-Cubed (R3): Rate, Robustness and Recovery - An Availability Benchmark Framework,” TR-2002-109, Sun Microsystems.
11. A. Ghosh, M. Schmid, “An Approach to Testing COTS Software for Robustness to Operating System Exceptions and Errors,” in *Proc. of International Symposium on Software Reliability Engineering*, 1999.
12. V. Sieh, K. Buchacker, “UMLinux — A Versatile SWIFI Tool” In *Proc. of Fourth European Dependable Computing Conference*, 2002, pp. 159-171.
13. H. Madeira *et al.*, “The OLAP and Data Warehousing Approaches for Analysis and Sharing of Results from Dependability Evaluation Experiments,” in *Proc. DSN-DCC 2003*, USA, June 2003.
14. Pintér G. *et al.*, “A Data Mining Approach to Identify Key Factors in Dependability Experiments,” in *Proc. EDCC-5*, pp 263-280, 2005.
15. C. Csallner, Y. Smaragdakis, “JCrasher: An automatic robustness tester for Java”, *Practice & Experience*, vol. 34, no. 11, Sep. 2004, pp. 1025-1050
16. SAF Test, SAF-conformance test suite, URL: <http://safstest.sourceforge.net/>

Modular Formal Analysis of the Central Guardian in the Time-Triggered Architecture¹

Holger Pfeifer^{*}, Friedrich W. von Henke

*Abtl. Künstliche Intelligenz
Fakultät für Informatik
Universität Ulm
D-89069 Ulm*

Abstract

The Time-Triggered Protocol TTP/C constitutes the core of the communication level of the Time-Triggered Architecture for dependable real-time systems. TTP/C ensures consistent data distribution, even in the presence of faults occurring to nodes or the communication channel. However, the protocol mechanisms of TTP/C rely on a rather optimistic fault hypothesis. Therefore, an independent component, the central guardian, employs static knowledge about the system to transform arbitrary node failures into failure modes that are covered by the fault hypothesis.

This paper presents a modular formal analysis of the communication properties of TTP/C based on the guardian approach. Through a hierarchy of formal models, we give a precise description of the arguments that support the desired correctness properties of TTP/C. First, requirements for correct communication are expressed on an abstract level. By stepwise refinement we show both that these abstract requirements are met under the optimistic fault hypothesis, and how the guardian model allows a broader class of node failures to be tolerated.

The models have been developed and mechanically checked using the specification and verification system PVS.

^{*} Corresponding author.

Email addresses: Holger.Pfeifer@uni-ulm.de (Holger Pfeifer),
friedrich.von-henke@uni-ulm.de (Friedrich W. von Henke).

¹ This research was supported by the European Commission under the IST project NEXT TTA (IST-2001-32111).

1 Introduction

The Time-Triggered Architecture (TTA) [1–3] is a distributed computer architecture for the implementation of highly dependable real-time systems. In particular, it targets embedded control applications, such as *by-wire* systems in the automotive or aerospace industry [4,5]. For these safety-critical systems fault tolerance is of utmost importance. The Time-Triggered Protocol TTP/C constitutes the core of the communication level of the Time-Triggered Architecture. It furnishes a number of important services, such as atomic broadcast, consistent membership and protection against faulty nodes, that facilitate the development of these kinds of fault-tolerant real-time applications. However, these protocol mechanisms rely on a rather optimistic fault hypothesis and assume that a fault is either a reception fault or a consistent send fault of some node [6]. In order to extend the class of faults that can be tolerated a special hardware component, the so-called *guardian*, is introduced [7]. A guardian is an autonomous unit that protects the shared communication network against faulty behaviour of nodes by supervising their output. The original bus topology of the communication network employed local bus guardians, which were placed between the nodes and the bus. In the more recent star topology, central guardians are used in the hub of each star. The guardian makes use of static knowledge available in a TTA-based system to transform arbitrary node failures into those that are covered by the optimistic fault hypothesis. For example, the time interval during which a given node is allowed to access the shared communication network is statically determined in a TTA system and known *a priori*. The guardian can hence control the correct timing of message transmissions by granting write access to the network only during a node’s pre-defined time slot.

The goal of this work is to formally model TTP/C guardians and analyse their fault tolerance properties. In particular, we aim at describing the benefits of the guardians by giving a precise specification of the assumptions on which the derivation of the properties is based. Formal analysis can provide an additional source of confidence in correct behaviour of a system, which is particularly important in the context of safety-critical systems. Several aspects of TTP/C and related protocols have therefore been formally modelled and analysed, including clock synchronisation [8], group membership [9–11], and the startup procedure [12,13]. A detailed overview of formal analysis work for the Time-Triggered Architecture is given by Rushby [14]. While so far the protocol algorithms of the time-triggered protocol have been the focus of the formal analyses cited above, we concentrate in this paper on the communication properties of TTP/C, thereby complementing and extending previous work.

To describe the behaviour and properties of the communication network and

the guardians we develop various formal models, which are organised in a hierarchical fashion. We start by specifying the desired correctness properties of the communication in an abstract form. Subsequently, in a process of stepwise refinement, more detail is added to this initial abstract model. On the next level of the hierarchy, we consider a TTP/C system without guardians. We show that in this case the strong, optimistic fault hypothesis is necessary to guarantee correct communication. Another model then introduces guardians and specifies their behaviour. At this level we demonstrate that the optimistic assumptions can be relaxed, which leads to a fault hypothesis that covers a broader class of faults.

The development of the models is in the spirit of, and builds on the work on modelling TTP-related aspects that has been carried out previously [8, 10, 15]. Specifically, it continues the use of the PVS specification and verification system [16] to both specify the model and the properties to be verified, and develop formal proofs that the model satisfies the stated properties. Previous work has demonstrated the suitability of PVS for this type of tasks. The formal models that are presented in this paper have been developed, and the proofs of their properties have been mechanically checked, with the PVS system.

The paper is organised as follows. In Section 2 we give a brief overview of the main aspects of the Time-Triggered Architecture. Section 3 describes the structure of the models and motivates their organisation. Details of the components of the formal models are elaborated in Section 4. Finally, we conclude in Section 5.

2 Brief Overview of the Time-Triggered Architecture

In this section we only briefly describe the main aspects of the Time-Triggered Architecture to the extent that is required for this paper. For more detailed presentations we refer to [3, 17, 18].

In a Time-Triggered Architecture system a set of *nodes* are interconnected by a real-time communication system. A node consists of the host computer, which runs the application software, and the communication controller, which accomplishes the time-triggered communication between different nodes. The nodes communicate via replicated shared media, the communication *channels*. There are two common physical interconnection topologies for TTA. Originally, the channels were replicated, passive buses, while in the more recent star topology the nodes are connected to replicated central star couplers, one for each of the two communication channels.

The distinguishing characteristic of time-triggered systems is that all system

activities are initiated by the passage of time [19]. The autonomous TTA communication system periodically executes a time-division multiple access (TDMA) schedule. Thus, access to the communication medium is divided into a series of intervals, called *slots*. Every node exclusively owns certain slots in which it is allowed to send messages via the communication network. The times of the periodic message sending actions are determined *a priori*, that is, at design time of the system. The send and receive instants are contained in a message schedule, the so-called *message descriptor list (MEDL)*. This scheduling table is static and stored at each communication controller. It thus provides common knowledge about message timing to all nodes. A complete cycle during which every node has access to the network exactly once is called a *TDMA round*.

Messages are used as a life-sign of the respective sender, and whenever a node receives a correct frame on at least one of the channels it considers the sender correct. Correctness of a frame is determined by each receiving node according to a set of criteria. A node considers a frame correct if it is well-timed, i. e. arrives within the boundaries of the TDMA slot, the physical signal obeys the line encoding rules, the frame passes a CRC check, and the sender and receiver agree on the distributed protocol state, the so-called *C-state*. One of the desired correctness properties of TTP/C is that all correct nodes always agree on whether or not a message is considered correct.

The Time-Triggered Protocol is designed to provide fault tolerance. In particular, the protocol has to ensure that non-faulty nodes receive consistent data despite the presence of possibly faulty nodes or a faulty communication channel. The provision of fault tolerance is based on a number of assumptions about the types, number, and frequency of faults. Altogether, these assumptions constitute the so-called *fault hypothesis*. The main assumption for the algorithms implemented in TTP/C is that a fault manifests itself as either a reception fault or a consistent send fault of some node [6]. In particular, the TTP/C services rely on transmission faults being consistent. That is, messages must be received correctly by either all non-faulty nodes or none. Moreover, nodes are assumed not to send messages outside their assigned slots. With respect to faults of the communication network, it is assumed that the channels cannot spontaneously create correct messages, and that messages are delivered either with some known bounded delay or never. With regard to the frequency and number of faults, TTP/C assumes that only one node becomes faulty during a TDMA round, and that there is at most one faulty node or one faulty channel at a time.

However, the Time-Triggered Architecture can tolerate a broader class of faults by intensively using the static knowledge that is present in the TDMA schedule. This allows to transform arbitrary failure modes of nodes into either send or receive faults that can be tolerated by the protocol. The guardians, which

are dedicated components of the communication system, monitor the temporal behaviour of the nodes. As the right to access the communication channels is statically determined, the guardians can bar a faulty node from sending a message outside its designated slots. Thus, timing failures of nodes are effectively transformed into send faults.

Moreover, guardians can protect against a particular class of *Byzantine* faults, the so-called *slightly-off-specification* (SOS) faults. A component is called SOS-faulty if it exhibits only marginally faulty behaviour that appears correct to some components, but faulty to others. A slightly-off-specification timing fault could occur if the transmission of a node terminates very close to the end of its scheduled transmission interval; thus, some receivers might accept the message while others might consider it mistimed. Because the duration of a particular transmission is known beforehand, the guardian can prevent such a cut-off scenario. A node must begin its transmission during a pre-defined period of time after the start of its slot, otherwise the guardian would terminate the right to access the communication network. Thus, the guardian can effectively prevent cut-off SOS faults, provided that the transmission interval is chosen long enough to ensure that a transmission fits the interval whenever it is started in time. Specifically, TTP/C guardians protect against SOS faults in the line encoding of frames at the physical layer, SOS timing faults, transmission of data outside the designated sending slots, masquerading of nodes, and transmission of non-agreed critical state information [7].

3 Bird's Eye View of the Formal Models

The overall goal of modelling the communication network is to provide a concise description of the arguments that support the following three main correctness properties of the TTP/C communication:

- *Validity:*
If a correct node transmits a correct frame, then all correct receivers accept the frame.
- *Agreement:*
If any correct node accepts a frame, then all correct receivers do.
- *Authenticity:*
A correct node accepts a frame only if it has been sent by the scheduled sending node of the given slot.

Once these properties are established, they can be exploited in subsequent analyses of protocol algorithms. This is preferable, since it is generally more feasible to base an analysis on properties of a supporting model or theory, rather than on the mere definitions of the model itself.

In order to facilitate the deduction, the formal proofs of these properties are decomposed into a series of smaller steps, and a hierarchy of corresponding models has been developed. Each of the single models focuses on a particular aspect of the communication. Altogether, we have identified the following four suitable model layers:

- General specification of the reception of frames.
- Channels without guardians, requiring a strong fault hypothesis.
- Channels with guardians, requiring only a weaker fault hypothesis.
- Different network topologies: local bus guardians and central guardians.

Each of the models contributes a small step towards proving the desired correctness properties. The steps themselves are each based on a set of assumptions or preconditions. Put in an abstract, and maybe also slightly over-simplified way, in each model layer i one establishes a theorem of the form

$$assumptions_i \Rightarrow properties_i$$

The idea is to design the different models in such a way that the properties on one level establish the assumptions on the next. Ultimately, the models are integrated and the reasoning is combined, yielding a chain of implications of roughly the following kind:

$$\begin{aligned}
 assumptions_0 &\Rightarrow properties_0 \\
 &= \text{ or } \Rightarrow \\
 assumptions_1 &\Rightarrow properties_1 \\
 &= \text{ or } \Rightarrow \\
 &assumptions_2 \Rightarrow \dots \Rightarrow properties_f
 \end{aligned}$$

The final properties, $properties_f$, correspond to the desired main correctness properties of the TTP/C communication as specified above, while the initial assumptions, $assumptions_0$, describe what constitutes the basic fault hypothesis.

We are going to briefly summarise the main aspects of the four model layers. At the bottom, the model describes the reception of frames by the nodes. Here, the various actions that nodes take in order to judge the correctness of the received frame are formalised. This amounts to considering the transmission time and the signal encoding of the frame, and the outcomes of the CRC check and the C-state agreement check, respectively [18]. The main correctness properties of the communication network are then expressed in terms of these notions. The assumptions of this model layer concern requirements about the functionality of the communication channels. In particular, they describe properties of the

frames that a channel transmits, such as signal encoding or delivery times, and reflect the hypothesis about possible faults of the communication network. In essence, this model establishes a proposition that informally reads as follows:

$$general_channel_properties \Rightarrow Validity \wedge Agreement \wedge Authenticity \quad (1)$$

On the next level, we model the transmission of frames through channels that are not equipped with guardians. The goal is then to derive the assumptions of the basic model, as covered by the expression *general_channel_properties*. However, in order to do so, a strong hypothesis on the types of possible faults of nodes is necessary. This strong fault hypothesis requires, for instance, that even a faulty node does not send data outside its sending slot, and nodes never send correct frames when they are not scheduled to do so. Using our informal notation, we can sketch the reasoning at this level as follows:

$$strong_fault_hypothesis \Rightarrow general_channel_properties \quad (2)$$

Guardians are employed to transform arbitrary node faults into faults that are covered by the strong fault model. Thus, the strong fault hypothesis can be replaced with weaker assumptions about the correct behaviour of the guardians. The functionality and the properties of the guardians are formally specified in the third model of the hierarchy, where the following fact is established:

$$weaker_fault_hyp. \wedge generic_guardian \Rightarrow general_channel_properties \quad (3)$$

Ideally, we would have liked to demonstrate directly that – together with the guardian properties – the weak form of the fault hypothesis implies the strong one. However, it turned out to be rather challenging to accomplish a formal proof for this fact and hence we had to revert to reasoning according to (3).

The model of the guardians is generic, as it does not, for instance, stipulate the type of guardian to be used in the communication network. The final level of our hierarchy models each of the two typical topologies of a TTP/C network: the bus topology and the star topology. In the former, each node of the network is equipped with its own local bus guardian, one for each channel, while in the latter the guardians are placed into the central star-coupling device of the channels. In this model layer we show that the properties of the guardians are independent from the choice of a particular topology, given that both the local bus guardians and the central guardians implement the same algorithms. Hence, we establish the following facts:

$$local_bus_guardian \Rightarrow generic_guardian \quad (4)$$

$$central_star_guardian \Rightarrow generic_guardian \quad (5)$$

The hierarchic arrangement of the models for the communication network allows for a concise description of the dependencies of the three main correctness properties. At the basic level the fundamental prerequisites are described that are necessary for the desired correctness properties to hold, while the subsequent levels express what must be assumed from the nodes and guardians, respectively, to satisfy these prerequisites. In particular, the treatment precisely explains the benefits of introducing guardians into the communication network.

4 Modular Formal Analysis of TTP/C Communication

In this section we present the main details of the formal models for the communication network according to the hierarchy that has been set out in the previous section. Although the formal models have been developed as a set of PVS modules (i.e., theories), the presentation is in the style of a mathematical transcription of these PVS modules. Similarly, we will explain the essential steps of the major proofs in an informal way; nevertheless, all proofs presented in this section have been developed and mechanically checked using the PVS theorem prover.

4.1 Modelling the Reception of Frames

We start the presentation of our hierarchy of models at the bottom level. This model provides a formalisation of how the communication of TTP/C essentially works. In particular, the actions taken by the nodes when sending or receiving frames are described. In addition, the desired correctness properties of TTP/C communication are stated at this level. The reference points of this formalisation are the TTP/C specification document [18] and the protocol developer's discussion of the fault assumptions in TTP/C [7]. In the sequel, we will refer to this model layer as the *ground model*.

In our model we consider a network with an undetermined but fixed number of communication channels. This is a generalisation of the TTA, which typically involves only two channels. In the formal model, we divide communication between nodes into three phases: the sending of frames by a sending node, the transmission of the frame on a channel, and the reception of the frame at the receiving nodes. To model the reception of frames we introduce a function $rcvd(n, c, r)$ to denote the frame a receiving node r receives in slot n on channel c . Similarly, $sent(n, c, p)$ denotes the frame that a node p has sent in slot n on channel c , while $transmit(n, c)$ models the frame that is transmitted on channel c in slot n . The ultimate goal of the ground model of our hierarchy

is to precisely state the relationship between these entities and to prove the correctness properties *Validity*, *Agreement*, and *Authenticity* explained in the previous section.

To this end, we first need to introduce notions expressing the checks a receiver carries out to determine the correctness of the frame received, and stating the conditions under which these checks are assumed, or required, to succeed. First of all, we formalise the notion of frame status. In TTP/C, frames can be *null* frames, *valid* frames, or *correct* frames. Frames that are neither null frames not valid, are called *invalid* frames, while valid frames that are not correct are called *incorrect*. TTP/C furthermore distinguishes tentative frames and frames that have other errors; the former relates to situations in the implicit acknowledgement process of nodes, while the latter refers to illegal mode change requests. These types of frames are, however, not considered in our model. The status of the frame received by node r on channel c in slot n will be denoted by $frame_status(n, c, r)$.

Frames are considered (syntactically) valid if the frame is transmitted during the receive window of the receiving node, no code violations are observed during the reception, and no other transmission was active within the receive window before the start of the frame. For a frame to be considered correct, it has to pass both the CRC check and the C-state agreement check [6].

Now we can formally state the desired correctness properties introduced in the previous section. The node scheduled to send in slot n is denoted $sender(n)$, while we use the (overloaded) notation \mathcal{NF}^n to denote both the set of non-faulty nodes and the non-faulty channels in slot n ; consequently, $r \in \mathcal{NF}^n$ and $c \in \mathcal{NF}^n$ indicate that node r and channel c are non-faulty in slot n .

Property 1 (Validity) *For all slots n , there exists a channel c such that if the sender of slot n sends a correct frame on c then all non-faulty nodes will receive this frame and assign the status correct to it:*

$$\begin{aligned} \exists c : sender(n) \in \mathcal{NF}^n \wedge sends_correct(n, c, sender(n)) \Rightarrow \\ \forall r \in \mathcal{NF}^n : rcvd(n, c, r) = sent(n, c, sender(n)) \wedge \\ frame_status(n, c, r) = correct \end{aligned}$$

Here, the predicate $sends_correct(n, c, sender(n))$ subsumes what is considered a correct sending action of a node: the sending node sends a non-null frame, does so at the specified time, the frame carries the correct C-state information and the physical signal obeys the line encoding rules.

Property 2 (Agreement) *All non-faulty nodes consistently assign the sta-*

thus correct to a frame received on a non-faulty channel c :

$$p \in \mathcal{NF}^n \wedge q \in \mathcal{NF}^n \wedge c \in \mathcal{NF}^n \Rightarrow \\ \text{frame_status}(n, c, p) = \text{correct} \Leftrightarrow \text{frame_status}(n, c, q) = \text{correct}$$

Property 3 (Authenticity) *A non-faulty node r assigns the frame status correct to a frame received on a non-faulty channel c only if it was sent by the scheduled sender of the slot:*

$$r \in \mathcal{NF}^n \wedge c \in \mathcal{NF}^n \wedge \text{frame_status}(n, c, r) = \text{correct} \Rightarrow \\ \text{rcvd}(n, c, r) = \text{sent}(n, c, \text{sender}(n))$$

In order to prove that these desired correctness properties hold for our model, several preconditions must be satisfied. These conditions concern the relationship between the frames sent by a sending node, transmitted through the channel, and received by the receivers, as expressed by the functions $\text{sent}(n, c, p)$, $\text{transmit}(n, c)$, and $\text{rcvd}(n, c, r)$, respectively. Therefore, we have to axiomatise the intended meaning of these functions. First of all, we formalise what is expected from a correct receiver. If a node r is non-faulty, we assume that it receives the frame that is transmitted on a given channel c :

$$r \in \mathcal{NF}^n \Rightarrow \text{rcvd}(n, c, r) = \text{transmit}(n, c) \quad (6)$$

However, even faulty nodes cannot receive other messages than those transmitted. Hence, nodes either receive whatever is transmitted on a channel, or nothing in the case of a reception fault:

$$\text{rcvd}(n, c, r) = \text{transmit}(n, c) \vee \text{rcvd}(n, c, r) = \text{null} \quad (7)$$

As for the sending nodes, we would like to model that non-faulty nodes send frames in their own sending slots, and remain silent otherwise. In some situations however, e. g. during the start-up of a TTP/C system or during the re-integration process of a node, the scheduled sender might not be fully integrated in the system and therefore, although being non-faulty, will not send at all in its sending slot. To also cope with these situations, we assume that in their sending slots non-faulty nodes either send a correct frame on all channels, or do not send any frame on any channel:

$$p = \text{sender}(n) \wedge p \in \mathcal{NF}^n \Rightarrow \\ (\forall c : \text{sends_correct}(n, c, p)) \vee (\forall c : \text{sent}(n, c, p) = \text{null}) \quad (8)$$

Next we need to constrain the behaviour of the channels. As we intend to examine both channels with and without guardians, we now state certain requirements that must be satisfied by either configuration in order to maintain the correctness properties.

First, we like to express that non-faulty channels deliver the frame sent by some node. However, a faulty node might try to send a frame on a channel outside its assigned sending slot, which could then interfere with the frame sent by the scheduled sending node. For our ground model, we do not want to constrain the behaviour of the channel in this case, but allow the channel to either transmit one of the frames sent by the interfering nodes, or block all transmissions, or transmit a corrupted frame.

Requirement 1 *A non-faulty channel either transmits the frame sent by some node p , or nothing, or a corrupted frame.*

$$c \in \mathcal{NF}^n \Rightarrow (\exists p : \text{transmit}(n, c) = \text{sent}(n, c, p)) \\ \vee \text{transmit}(n, c) = \text{null} \vee \text{corrupted}(\text{transmit}(n, c))$$

However, this requirement is not sufficient, as it allows trivial, and rather useless, solutions such as channels that never transmit anything. In order to exclude these unwanted cases, we require that a non-faulty channel must transmit the frame of the scheduled sending node in situations where no other node interferes. We use the predicate $\text{single_access}(n, c)$ to express that there is at most one node sending on channel c in slot n . Technically, this expression is an abstract parameter of our model; its interpretation depends on the concrete implementation of the channels, and it will be defined later in the subsequent refining model layers.

Requirement 2 *If the scheduled sender exclusively accesses the channel and sends a correct frame, then this frame is transmitted by the channel.*

$$c \in \mathcal{NF}^n \wedge \text{single_access}(n, c) \wedge \text{sends_correct}(n, c, \text{sender}(n)) \Rightarrow \\ \text{transmit}(n, c) = \text{sent}(n, c, \text{sender}(n))$$

The last of the basic requirements for non-faulty channels accounts for the fact that channels are passive entities and thus cannot generate frames by themselves. Here, the expression $\text{sends}(n, c, p)$ is an abbreviation for $\text{sent}(n, c, p) \neq \text{null}$.

Requirement 3 *Channels can only transmit what has been sent by some*

node.

$$\text{transmit}(n, c) \neq \text{null} \Rightarrow \exists p : \text{sends}(n, c, p)$$

We are now going to derive the proofs of the three main correctness properties for the TTP/C communication. However, as we will see, the assumptions and requirements listed so far are not sufficient to allow such a derivation. Consequently, we need to further constrain the behaviour of both the channels and the sending nodes in order to achieve correct communication. We will introduce the additional requirements as they become necessary in the derivation of the proofs.

4.1.1 Proof of Validity

The proposition *Validity* states two aspects of the reception of a correct frame sent by the scheduled sender in a given slot: first, all non-faulty receivers must receive the frame sent by the sender, and, second, all of them must accept this frame. With respect to the first part, we can derive from (6) that all non-faulty receivers receive the frame that is transmitted by the channel c . Furthermore, Req. 2 states that a non-faulty channel transmits the frame sent by the sender, provided that there is no other node accessing the channel. This latter clause gives rise to another requirement on the communication network:

Requirement 4 *In every slot, there is at least one non-faulty channel that is accessed by at most one node.*

$$\exists c \in \mathcal{NF}^n : \text{single_access}(n, c)$$

Note that this is a rather strong requirement, and one that is impossible to satisfy for a channel without further measures, because it requires a certain behaviour of faulty nodes, which is outside the control of a channel. As we will see in the subsequent sections, the treatment of this requirement is one that distinguishes the communication model with guardians from one without. With this requirement we can prove the first part of *Validity*, i.e., that all correct nodes receive the frame sent by the sender.

As for the second part, we need to demonstrate that all correct receivers assign the status *correct* to the frame, that is, that they see a non-null, valid frame that passes both the CRC check the C-state agreement check. Non-emptiness of the frame can be proved from the fact that a correct sender sends a correct, and thus non-null, frame. A correct frame will always be transmitted by a non-faulty channel due to Req. 2, and Req. 4 ensures that such a non-faulty channel does indeed exist. Concerning the validity of the frame we have to consider the transmission timing of the frame and its signal encoding on the physical

layer. The latter is given by the same line of arguments as we demonstrated that the frame is not a *null*-frame: a correct sender sends a correct frame, which includes a correct signal encoding, and a correct channel will transmit this frame.

Next, we focus on the transmission timing of the frame. In order for the frame to be received by a non-faulty receiver within its receive window, the sending node and the receiver must be synchronised. Moreover, the values defining the nominal sending time of a frame and the start and end of the receive window, respectively, must be chosen such that the possible slight differences among the readings of the nodes' local clocks allow for the receivers to open their receive windows at "the right time". If we presuppose that non-faultiness of nodes encompasses that they are synchronised and that the window timing parameters are set correctly, it suffices to show that the sender sends its frame in time, and that the channel has a transmission delay that is bounded by some given bound \hat{d} . The first is given by the fact that the sender sends a correct frame, and therefore the transmission time is correct. The second, however, must be stated as another requirement on the correct behaviour of a channel:

Requirement 5 *The transmission time of a correct frame on a non-faulty channel does not deviate from the sending time by more than some bounded delay d .*

$$c \in \mathcal{NF}^n \wedge \text{sends_correct}(n, c, p) \wedge \text{single_access}(n, c) \Rightarrow \\ \exists d : d \leq \hat{d} \wedge \text{transmission_time}(f') = \text{send_time}(f) + d \\ \text{where } p = \text{sender}(n), f = \text{sent}(n, c, p), f' = \text{transmit}(n, c)$$

The last characteristic of a valid frame is unique transmission, which is ensured by Req. 4.

So far we have shown that the received frame is considered valid by non-faulty receivers. We are thus left to examine the CRC check and the C-state agreement check. As for the first, an incorrect CRC checksum is used to signal transmission faults. As the channel c under consideration is a non-faulty one, it is reasonable to assume that this includes the fact that no transmission fault occurs on c . Therefore we can conclude that a frame received by a non-faulty receiver on a non-faulty channel will pass the CRC check.

Finally, we consider the C-state agreement check. For the frame to pass the check, the C-state encoded in the frame has to correspond to the receiver's C-state. For this to be the case, two things must be ensured: first, the sender and the receiver must have equal C-states, and, second, the sender provides a correct encoding of its C-state in the frame. The first part corresponds to the functionality of the clique avoidance mechanism of the TTP/C group

membership algorithm, which is responsible for maintaining a single clique of nodes during system operation, that is, a single group of nodes that has equal C-states. We abstract from this protocol property by assuming that our notion of non-faultiness of nodes includes that two non-faulty nodes belong to the same (single) clique and thus have common C-states.

The second part, however, gives rise to another requirement on the behaviour of a correct channel.

Requirement 6 *A non-faulty channel transmits a frame with a correct signal encoding only if the frame sent provides a correct encoding of the sender's C-state.*

$$c \in \mathcal{NF}^n \wedge \text{transmit}(n, c) = \text{sent}(n, c, p) \wedge \text{sends}(n, c, p) \wedge \\ \text{signal_encoding_OK}(\text{transmit}(n, c)) \Rightarrow \\ \text{cstate_encoding_OK}(n, \text{sent}(n, c, p), p)$$

We can summarise that with the requirements introduced above the *Validity* property can be derived.

4.1.2 Proof of Agreement

To prove *Agreement*, we have to demonstrate that if some non-faulty receiver considers a received frame correct, then all non-faulty receivers do so. To establish this property we have to prove the same six characteristics of the received frame as for *Validity*, that is, reception of a non-null frame, the three properties of valid frames, and the two correctness checks. Each of these six cases can be proved using the same requirements as the proof of *Validity*. The structure of the proof of the agreement property is very similar to that of the *Validity* property; we therefore omit a detailed description.

4.1.3 Proof of Authenticity

For *Authenticity* we are required to show that if a frame is considered correct by a correct receiver then this node has in fact received the frame sent by the scheduled sender of the slot. In order to derive this fact we note that since the receiver considers the frame correct, we know that the six characteristics that define correct frames hold. This implies, for instance, that the receiver has detected a non-null frame. As channels only broadcast frames that have actually been sent by some node, cf. Req. 3, we know that there is an originator of the frame and that the receivers have received the frame sent by this sending node, say p . Hence, we only need to prove that this node p is in fact the

scheduled sender of the current slot. However, the facts established so far are not sufficient to do so, and hence we need to introduce one final requirement on the behaviour of channels.

Requirement 7 *A non-faulty channel transmits a correctly sent frame only if it originates from the scheduled sender of the given slot.*

$$c \in \mathcal{NF}^n \wedge \text{sends_correct}(n, c, p) \wedge \text{transmit}(n, c) = \text{sent}(n, c, p) \Rightarrow \\ p = \text{sender}(n)$$

This requirement, together with the precondition that the received frame is considered correct by the receiving node, enables us to prove that the originator of the frame is indeed the scheduled sender of the given slot.

This concludes the derivation of the three desired correctness properties for the communication of TTP/C and the requirements they are based on. In the following two sections we will describe under which fault hypotheses these requirements can be met, both for a scenario with and without bus guardians.

4.2 Strong TTP/C Fault Hypothesis

In the previous section we have described a formalisation of the reception of frames by a node and have stated seven requirements that must be satisfied by the sending nodes and the channels in order to establish the desired correctness properties. In this section we are now going to give a formal model for sending nodes and channels and examine how the requirements stated above can be met. First, we consider the scenario where the channels are simple passive entities that broadcast the frames sent by a sender without further mechanisms, before, in the next section, we analyse a refinement of this model that incorporates bus guardians.

The ground model presented in the previous section expresses certain required properties of the entities $\text{sent}(n, c, p)$ and $\text{transmit}(n, c)$, which model the behaviour of the sending nodes and the channel, respectively. In a technical sense, these entities are parameters of the model. We now give an interpretation to these parameters for a network without guardians and show that the general requirements are satisfied for these interpretations.

First, we give a definition of the predicate $\text{single_access}(n, c)$. This predicate is intended to model the case where only the scheduled sender sends a frame, and no other node interferes. Hence, we define it as true if there are no two

different nodes that send a non-null frame on the channel in the same slot:

$$single_access(n, c) := \forall p, q : sends(n, c, p) \wedge sends(n, c, q) \Rightarrow p = q \quad (9)$$

The interpretation of $sent(n, c, p)$ and $transmit(n, c)$ is given in an axiomatic style, and the set of axioms essentially constitutes the fault hypothesis of the guardian-free setting. In this setting, we cannot say anything about the frame transmitted by a channel other than that it depends on what is sent by the sending nodes. This is in contrast to the scenario with guardians, where we can, for instance, express that a guardian will not broadcast a frame if it does not originate from the scheduled sender.

Hypothesis 1 *A non-faulty channel without a guardian will transmit a frame sent by a node p if no other node accesses the channel in the given slot n .*

$$c \in \mathcal{NF}^n \wedge sends(n, c, p) \wedge (\neg \exists q : q \neq p \wedge sends(n, c, q)) \Rightarrow \\ transmit(n, c) = sent(n, c, p)$$

This hypothesis is sufficient to prove Req. 2 of the ground model; to see this, note that with the definition of $single_access$, the premise of Req. 2 implies that of Hyp. 1.

In order to prove Req. 1, we must assume that a channel can only transmit a non-null frame if it has been sent by some node:

Hypothesis 2 *If a channel broadcasts a non-null frame, then there is a corresponding node that has sent this frame.*

$$transmit(n, c) \neq null \Rightarrow \exists p : sends(n, c, p)$$

With this assumption we can now prove Req. 1 of the ground model: either the frame transmitted on a channel is a null frame, or if it is not, then by Hyp. 2 there is a sending node p , and the channel transmits the frame sent by p according to Hyp. 1.

Note that Hyp. 2 is actually identical to Req. 3 of the ground model. At this level, we cannot further constrain the behaviour of the channels more than what is expressed by Hyp. 1; consequently, some of the requirements of the ground model have to be restated as hypotheses on the channels for the guardian-free case. This is also true for Req. 5, which constrains the possible delay in the delivery of a frame on a non-faulty channel:

Hypothesis 3 *The delivery time of a frame on a non-faulty channel does not*

deviate from the transmission time by more than some bounded delay d .

$$\begin{aligned}
& c \in \mathcal{NF}^n \wedge \text{sends}(n, c, p) \wedge f' \neq \text{null} \wedge \neg \text{corrupted}(f') \Rightarrow \\
& \quad \exists d : d \leq \hat{d} \wedge \text{transmission_time}(f') = \text{sending_time}(f) + d \\
& \text{where } f = \text{sent}(n, c, p), f' = \text{transmit}(n, c)
\end{aligned}$$

Thus, Req. 3 and Req. 5 are trivially satisfied by our model. Note, however, that the corresponding assumptions are by no means just inadmissible simplifications of the matter. On the contrary, these hypotheses are direct formalisations of the strong fault hypothesis of the “raw” TTP/C protocol [7].

We proceed by extending our model in order to also derive the remaining three requirements of the ground model.

Considering Req. 6, we need to ensure that a non-faulty channel only transmits frames that contain a correct encoding of the sender’s C-state. Since in the guardian-free setting, the channels will transmit whatever is sent by the sending node, the responsibility for providing a correct C-state encoding is with the sender. Note that this must be true not only for the scheduled sender, but extends to all nodes, even faulty ones, and hence is a rather strong assumption.

Hypothesis 4 *Frames sent must contain a correct encoding of the sender’s C-state.*

$$\text{sends}(n, c, p) \Rightarrow \text{cstate_encoding_OK}(n, \text{sent}(n, c, p), p)$$

Requirement 7 states that correct frames must only be sent by the scheduled sender of a given slot. However, for the guardian-free case a channel cannot prevent other nodes from sending. Hence, in order to prove this property for this model, we need to introduce a corresponding hypothesis and assume that the behaviour of the sending nodes is in compliance with the sending schedule.

Hypothesis 5 *Correct frames must only be sent by the scheduled sender of a given slot.*

$$\text{sends_correct}(n, c, p) \Rightarrow p = \text{sender}(n)$$

We are left to prove Req. 4, which states that in every slot there exists at least one non-faulty channel that is not accessed by more than one sending node. To prove this fact we have to make a series of assumptions about the number and behaviour of faulty nodes and channels. First of all, we need to assume that a non-faulty channel exists at all times.

Hypothesis 6 *In every slot, there is at least one non-faulty channel.*

$$\exists c \in \mathcal{NF}^n$$

TTP/C is based on a *single fault assumption*, i. e. at any given time there is at most one faulty component in the network. Consequently, there cannot be more than one faulty node present in any given slot.

Hypothesis 7 *There is at most one faulty node in every slot.*

$$p \notin \mathcal{NF}^n \wedge q \notin \mathcal{NF}^n \Rightarrow p = q$$

In deriving Req. 4, we first consider the case where there is no faulty node. By Hyp. 6 we know that there exists a non-faulty channel c . To establish Req. 4 we must therefore show that at most one node sends a frame on c . Since the scheduled sender of the given slot is allowed to send a frame, we must ensure that no other node can send. We can establish this fact if we assume that a *non-faulty* node does not send anything outside its designated sending slots, which is a reasonable assumption to make.

$$p \neq \text{sender}(n) \wedge p \in \mathcal{NF}^n \Rightarrow \neg \text{sends}(n, c, p) \quad (10)$$

Now suppose that there is a faulty node, p say. If p is the scheduled sender of the given slot we are done, because then all other nodes are non-faulty, by Hyp. 7, and do not send a frame, see Hyp. 10. Therefore consider the case where the faulty node p is not the current sender. In order to prevent a collision on the channel we must require that p , even if it is faulty, does not send.

Hypothesis 8 *Nodes other than the sender of a slot, including faulty ones, will not send data on every non-faulty channel outside their assigned sending slots.*

$$p \neq \text{sender}(n) \Rightarrow \neg \forall c \in \mathcal{NF}^n : \text{sends}(n, c, p)$$

Note that the hypothesis as stated only requires that p does not send on at least one of the non-faulty channels. Again, this is a strong hypothesis, as it constrains the behaviour even of *faulty* nodes.

This completes our derivation of the seven requirements of the ground model for a communication network without guardians. In order to establish the requirements we have stated a series of hypotheses. Besides describing the intended behaviour of correct nodes and channels, these assumptions directly reflect the strong fault hypothesis of the “raw” TTP/C protocol [7]. We have shown that this fault model is sufficient to prove the requirements of the

ground model, and thus established the desired correctness properties for the communication in a TTP/C network. What makes this set of hypotheses strong or optimistic is the fact that assumptions are not restricted to non-faulty nodes, but also encompass faulty ones, cf. Hyp. 4, 5, and 8. In the following section, these hypotheses will be replaced with weaker ones about the behaviour of *non-faulty* guardians.

4.3 Guardians

In the scenario described in the previous section, where a channel transmits a frame whenever there is no concurrent access to it, strong assumptions about the behaviour of the sending nodes have to be made in order to satisfy the requirements stated in the ground model. In particular, some of the assumptions even concern the behaviour of faulty nodes, such as that sending nodes always provide a correct C-state in the frame, or that correct frames are sent only by the scheduled sender of a slot. Whenever one relies on a certain benignity of faults one has to examine how well the fault assumptions are covered by the system. If such an analysis is difficult, or leads to the result that the probability of a fault being outside of the scope of the assumed fault hypothesis is not negligible, it is advisable to aim to eliminate, or at least weaken, assumptions about the behaviour of faulty components. To this end, guardians are used in the Time-Triggered Architecture to avoid certain fault scenarios, such as, for instance, faulty nodes accessing the bus outside their assigned slots.

In this section we describe the formalisation of abstract guardian components. The formalisation is abstract in the sense that it does not restrict the kind of the guardian and the topology of the communication network; we will show in the subsequent section how this abstract model can be refined either to a bus topology, where each node has its own local guardians, or to a star topology with central bus guardians.

We state a number of hypotheses on the expected behaviour of a non-faulty guardian and show that they are sufficient to prove the requirements of the ground model. Thus, the desired correctness properties for the communication of TTP/C are satisfied for a communication network with guardians.

In our model, we use $g(c)$ to denote the guardian of channel c . We think of a guardian as having incoming links from each of the nodes of the network, and corresponding outgoing links. The task of a guardian is to receive the frames sent by the nodes, analyse them and relay them to the other nodes according to certain rules. Obviously, these rules would prescribe, among other things, that only the frame of the scheduled sender of a slot is relayed. To describe the functionality of a guardian we use a function $relay(n, g(c), p)$ that denotes

the frame the guardian $g(c)$ relays from node p in slot n .

By the following hypotheses we describe what is expected from a non-faulty guardian. To distinguish the guardian hypotheses from the ones presented in the previous section, they are labelled with capital letters instead of numbers. First, if the scheduled sender of a slot sends a correct frame, then the guardian should relay this frame:

Hypothesis A *If the scheduled sender of a slot sends a correct frame, then a correct guardian relays this frame.*

$$p = \text{sender}(n) \wedge g(c) \in \mathcal{NF}^n \wedge \text{sends_correct}(n, c, p) \Rightarrow \\ \text{relay}(n, g(c), p) = \text{sent}(n, c, p)$$

Conversely, frames of nodes other than the scheduled sender must not be relayed:

Hypothesis B *A non-faulty guardian must not relay frames of nodes other than the scheduled sender.*

$$p \neq \text{sender}(n) \wedge g(c) \in \mathcal{NF}^n \Rightarrow \text{relay}(n, g(c), p) = \text{null}$$

In addition to this basic functionality of supervising the correct message schedule, the guardian performs several other analyses in order to prevent fault propagation and possible SOS faults. First, if a sending node does not start to send its frame within the nominal sending window, the guardian closes the window with the effect that a null frame is relayed.

Hypothesis C *A non-faulty guardian will relay a frame only if it is being sent in time.*

$$p = \text{sender}(n) \wedge g(c) \in \mathcal{NF}^n \wedge \neg \text{sending_time_OK}(n, \text{sent}(n, c, p), p) \Rightarrow \\ \text{relay}(n, g(c), p) = \text{null}$$

Furthermore, if the signal encoding of the frame sent by a node violates the coding rules such that the guardian cannot decode the signal, it will end the broadcast of the frame prematurely, thus corrupting the frame.

Hypothesis D *A non-faulty guardian will corrupt a frame if the signal encoding of the frame violates the coding rules.*

$$p = \text{sender}(n) \wedge g(c) \in \mathcal{NF}^n \wedge \neg \text{signal_encoding_OK}(\text{sent}(n, c, p)) \Rightarrow \\ \text{corrupted}(\text{relay}(n, g(c), p))$$

Finally, if the C-state encoded in a frame does not correspond to the guardian's own C-state, then the guardian aborts the transmission of the frame, and the relayed frame will be corrupted. This serves, for example, to protect a node that is about to integrate into the cluster against so-called *masquerading* nodes that provide an incorrect MEDL position within the C-state.

Hypothesis E *A non-faulty guardian will corrupt a frame if the C-state encoded in the frame does not correspond to the guardian's own C-state.*

$$\begin{aligned}
&g(c) \in \mathcal{NF}^n \wedge \neg \text{cstate_encoding_OK}(n, \text{sent}(n, c, p), p) \\
&\quad \Rightarrow \text{corrupted}(\text{relay}(n, g(c), p)) \\
&\text{where } p = \text{sender}(n)
\end{aligned}$$

These hypotheses describe the supervising functionality of a guardian. In addition, a non-faulty guardian is expected to behave in a reasonable way. First, guardians are assumed to be passive entities in the sense that they can only relay frames that have actually been sent by some node. In other words, guardians cannot generate valid frames by themselves.

Hypothesis F *Guardians are passive and can only relay frames that have actually been sent by some node.*

$$\text{relay}(n, g(c), p) \neq \text{null} \Rightarrow \text{sends}(n, c, p)$$

The next assumption on the functionality of a guardian concerns the timing behaviour. In order to fulfil Req. 5 of the ground model we must assume that a guardian delivers a relayed frame with a bounded delay.

Hypothesis G *A non-faulty guardian relays a frame with a bounded delay.*

$$\begin{aligned}
&g(c) \in \mathcal{NF}^n \wedge \text{sends}(n, c, p) \wedge f' \neq \text{null} \Rightarrow \\
&\quad \exists d : d \leq \hat{d} \wedge \text{transmission_time}(f') = \text{sending_time}(f) + d \\
&\text{where } f = \text{sent}(n, c, p), f' = \text{relay}(n, g(c), p)
\end{aligned}$$

The final two assumptions concern the number and kinds of possible faults of the guardians. First, we assume that for all slots the guardian of at least one of the channels is non-faulty.

Hypothesis H *For every slot n , there is at least one channel with a non-faulty guardian.*

$$\exists c : g(c) \in \mathcal{NF}^n$$

A faulty guardian may fail only in such a way that it delays the delivery of a frame for an arbitrary amount of time and thus effectively does not relay any non-null frame in the given slot n .

Hypothesis I *A faulty guardian fails silently and does not relay any frame.*

$$g(c) \notin \mathcal{NF}^n \Rightarrow \text{relay}(n, g(c), p) = \text{null}$$

Some of the requirements of the ground model involve the abstract predicate $\text{single_access}(n, c)$ and we must hence give an interpretation to this predicate for a communication network with guardians. Since the guardians are intended to just prevent the simultaneous access of a channel by two different nodes, we define this predicate to be always true:

$$\text{single_access}(n, c) := \text{true} \quad (11)$$

Finally, we say that a channel is non-faulty if its corresponding guardian is.

$$c \in \mathcal{NF}^n := g(c) \in \mathcal{NF}^n \quad (12)$$

To complete the formalisation of the guardian model, we need to define what is meant by the frame a channel broadcasts, i. e. we require a definition of the function $\text{transmit}(n, c)$. Obviously, this function definition must reflect the frame that a guardian relays for some node p . On the other hand, there cannot be frames from more than one node be transmitted per slot. Consequently, we say that a frame is transmitted on a channel c if there is a node p such that the guardian $g(c)$ of channel c relays that frame for p , and does not relay any frame for all nodes other than p . The technical definition of $\text{transmit}(n, c)$ proceeds in two steps. First, we define a predicate $\text{unique}_c^n(f)$ to be true, if in slot n the guardian of channel c relays frame f for some node, but relays no frames for any other node:

$$\begin{aligned} \text{unique}_c^n(f) := \exists p : \text{relay}(n, g(c), p) = f \wedge \\ \forall q : q \neq p \Rightarrow \text{relay}(n, g(c), q) = \text{null} \end{aligned} \quad (13)$$

For the definition of $\text{transmit}(n, c)$ we use Hilbert's choice operator ϵ , where $\epsilon(S)$ denotes some arbitrarily chosen element from a given set S . Here, the set S consists of those frames f for which the predicate $\text{unique}_c^n(f)$ is true. Obviously, this set can contain at most one frame; consequently, if the set is non-empty, simply this unique frame is chosen. In the other case where the set is empty, i. e. no frame satisfies the *unique*-predicate, the ϵ -operator returns

an arbitrary frame, for which no special properties can be deduced.

$$\text{transmit}(n, c) := \epsilon(\text{unique}_c^n) \quad (14)$$

In the remainder of this section we present the arguments that show that this definition of $\text{transmit}(n, c)$ and the hypotheses stated for a guardian are sufficient to satisfy the requirements of the ground model. To this end, we state two properties of $\text{transmit}(n, c)$. First note that a non-faulty guardian either transmits the frame sent by the scheduled sender of a given slot, or a corrupted frame, or a null-frame.

Proposition 1 *A non-faulty guardian either transmits the frame of the scheduled sender, or a corrupted frame, or a null-frame.*

$$c \in \mathcal{NF}^n \Rightarrow \text{transmit}(n, c) = \text{sent}(n, c, \text{sender}(n)) \vee \\ \text{corrupted}(\text{transmit}(n, c)) \vee \text{transmit}(n, c) = \text{null}$$

If in addition we know that the scheduled sender of a slot sends a correct frame, then the guardian indeed broadcasts this frame.

Proposition 2 *A non-faulty guardian transmits a correct frame if it is sent by the scheduled sender of slot n .*

$$c \in \mathcal{NF}^n \wedge \text{sends_correct}(n, c, \text{sender}(n)) \Rightarrow \\ \text{transmit}(n, c) = \text{sent}(n, c, \text{sender}(n))$$

We briefly sketch the proofs of these properties. First note that, according to Hyp. B, a non-faulty guardian does not relay a frame for nodes other than the scheduled sender of the given slot. Moreover, since guardians will not produce frames by themselves, see Hyp. F, the only frame that is relayed by a non-faulty guardian is the one sent by the scheduled sender of the slot. Hence, this frame satisfies the *unique*-predicate, and therefore $\text{transmit}(n, c)$ equals the frame that is relayed by the guardian for the scheduled sender of the current slot. Proposition 1 holds because, depending on whether or not the scheduled sender sends a correct frame, the guardian either relays this frame according to Hyp. A, or blocks or corrupts the frame following Hyp. C, D, or E.

The second proposition is a specialisation of the first, where we know that the scheduled sender sends a correct frame. In this case, the guardian relays this frame and by the same reasoning as above this frame is transmitted on the channel.

These two propositions provide the connection between the requirements of the ground model, which are stated in terms of the expression *transmit*, and the hypotheses of the guardian model, which are very similar, but involve the *relay* forms. To derive the general requirements, we see that Req. 1 of the ground model directly follows from the first of the propositions above, while the second proposition implies Req. 2. Requirement 3 can be proved from the similar assumption that guardians do not send frames by themselves, see Hyp. F. Requirement 4 follows from the assumption that there always exists a non-faulty guardian, see Hyp. H, and observing that the predicate *single_access*(n, c) is always true. Requirement 5 on the bounded delay of transmissions follows from Hyp. G, while the assumption concerning the encoded C-state of a frame, Hyp. E, is used to prove Req. 6. Finally, Req. 7 follows from the combination of the two propositions above.

Having shown that all of the requirements of the ground model are satisfied in the guardian model, we can deduce that the three correctness properties *Validity*, *Agreement* and *Authenticity* hold for the guardian model. In comparison to the model without guardians, weaker hypotheses are sufficient to prove the requirements. In particular, we do no longer need to make any assumptions about the behaviour of faulty nodes, thus a broader class of node faults can be tolerated by a TTP/C network using guardians.

4.4 Local vs. Central Guardians

In this section we briefly discuss how the guardian model described above can be applied to both an interconnection network with a star topology and one with a bus topology. In the former, central guardians are used, which are usually located at the centre of each communication channel, i.e. at the star coupler. Thus, the above guardian model can be directly matched to this setup, since the denotation $g(c)$ appropriately models the central guardian device at the star coupler of channel c .

In a connection network that uses the bus topology every node is equipped with its own local guardian, typically one for each channel. Therefore, one would rather use a function lbg to denote particular guardian devices, such that $lbg(p, c)$ is the local bus guardian of node p for channel c . Nevertheless, the functionality of the bus guardians can be described in the same way as in the star-topology model by formalising assumptions about the frames relayed by the local bus guardians, as expressed by the function $relay(n, lbg(p, c), p)$. In order to use the abstract guardian model of the previous section we only need to combine the local bus guardians of all nodes that supervise a particular channel c to one logical entity. Thus, the expression $g(c)$ would denote a function that yields for a given node its local bus guardian that controls

channel c . Formally:

$$g(c) := \lambda p : lbg(p, c) \tag{15}$$

Consequently, the system of local bus guardians at channel c is considered non-faulty, if for all nodes p the local bus guardian $g(c)(p)$ is non-faulty:

$$g(c) \in \mathcal{NF}^n \Leftrightarrow \forall p : lbg(p, c) \in \mathcal{NF}^n \tag{16}$$

To summarise, the abstract guardian model can be arranged in a way that the formalisations of guardians for both a star-based topology and a bus topology can be derived as an instance of this model. The details are, however, mainly of a technical nature and do not provide any further conceptual insight; therefore, they are omitted here. At the bottom line we can state that, as long as the same algorithms and supervising functions are implemented in either guardian type, both the local bus guardians and the central guardians of a star coupler provide the functionality to satisfy the requirements stated in the ground model and thus ensure that the main correctness properties for the communication of TTP/C hold.

5 Conclusions

The goal of formally analysing aspects of the Time-Triggered Architecture is to provide mathematically substantiated arguments that architecture and algorithms provide certain services and satisfy certain critical properties. This is to support the claims that the architecture meets the high reliability requirements of safety-critical applications in the automotive or aerospace domain.

In this regard we have presented a formal analysis of the guardian-based communication of TTP/C. We have developed a series of formal models of the interconnection network that are hierarchically structured and formalise different aspects of the communication of TTP/C nodes at various levels. The ground level provides a precise specification of the desired correctness properties of the TTP/C communication. It states several requirements that must be satisfied for the channels in order to guarantee that the correctness properties hold. These requirements serve as an interface of the model. In a process of stepwise refinement we have proved the validity of these properties for TTP/C by showing that the interface requirements hold for the refined model layers. The organisation of the model hierarchy not only facilitates the formal proof by dividing it into manageable steps. It also reflects the structure of what constitutes the Time-Triggered Protocol, viz. the communication controllers of the nodes, and the guardians. The former provide the fault-tolerant protocol

services on the basis of strong fault assumptions, which, in turn, are guaranteed by the guardians. Thus, one of the benefits of our formal analysis is that the formal models yield a concise formal description of the respective purposes and dependencies of these components, and precisely state the assumptions about the behaviour of a guardian, which previously had been stated only informally [7].

One of the characteristics of the formal models is their abstract nature. Abstraction is a fundamental prerequisite for the feasibility of formal analysis, as it allows for both structuring the models by providing abstract interfaces and hiding details unnecessary or irrelevant for the formal analysis and the demonstration of critical properties. An adequate structure of formal models allows one to concentrate on particular aspects of a TTA system, such as the behaviour of the guardians in the communication network. Different items can then be analysed separately from each other, assuming certain properties of other models where necessary. Moreover, abstract interfaces of the models also provide a certain degree of genericity, which enables one to express the commonalities of a range of designs in a coherent way. For instance, one of the model layers provides a generic treatment of the guardians. The model can then be refined to either a central guardian-based view, or to a model for local bus guardians, thereby covering the two typical network topologies of a TTA system.

The formal models presented in this paper have been developed with the specification and verification system PVS, and all proofs have been mechanically checked using PVS's theorem prover. Although the individual proofs of most of the properties and facts are relatively simple and straightforward, the use of a mechanical proof assistant has been found very valuable. One of the difficulties in developing formal models and proofs is to keep track of all details and the dependencies of the various properties. PVS is particularly useful for such tasks, as it does not only check the proofs provided for the claimed properties, but also provides bookkeeping functions to ensure that there are no gaps in the chain of arguments for a given fact. Moreover, if changes are made to a formal model, PVS requires all proofs of properties that depend on the changed model to be re-run. Thus, if changes cause proofs to be no longer valid, these will not go undetected.

A mechanism that has been found particularly useful is PVS's support of *theory assumptions*. In a PVS theory one declares the relevant entities of a formal model, and states – and then proves – the properties these entities have. Theories can be parameterised, and one can state certain assumptions about concrete interpretations of these parameters. The properties within such a parameterised theory are then based on these assumptions. If such a theory is instantiated, that is, the parameters are given concrete interpretations, PVS automatically creates proof obligations that require to show that the

stated assumptions indeed hold for the given interpretations. We have employed this mechanism for specifying our ground model of the general reception of frames. This model is parameterised with the entities describing the sending of frames by nodes, $sent(n, c, p)$, or the transmission of frames by a channel, $transmit(n, c)$, among others. The general requirements described in detail in Section 4.1 are expressed as PVS assumptions on these parameters. The desired correctness properties, such as *Validity* or *Agreement*, are proved relative to these assumptions. The formal models on the higher hierarchy levels that describe the strong fault model and the guardian model, cf. Sections 4.2 and 4.3, respectively, instantiate the ground model. Thus, PVS generates proof obligations that correspond to showing that the general requirements of the ground model are valid for the provided interpretations of $sent(n, c, p)$ or $transmit(n, c)$. This way, PVS provides support to ensure that eventually all claimed facts are indeed proved.

The analysis of the properties of the communication network of TTA has supported the claim that the functionality of the guardians ensures that arbitrary node failures are converted into fault modes the TTP/C protocol algorithms can tolerate. Thus, the strong fault hypothesis of TTP/C can be replaced by a weaker, minimal fault hypothesis on the correct behaviour of the guardians, which has two direct advantages. First, applications of TTA can rely on the architecture to tolerate a broad class of faults, and, second, protocol algorithms of TTP/C can be designed for and analysed under the strong fault model, which allows for simpler algorithms and significantly facilitates formal analysis.

References

- [1] H. Kopetz, The Time-Triggered Approach to Real-Time System Design, in: B. Randell, J.-C. Laprie, H. Kopetz, B. Littlewood (Eds.), Predictably Dependable Computing Systems, Springer-Verlag, 1995.
- [2] H. Kopetz, The Time-Triggered Architecture, in: Proc. 1st Intl. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC), 1998, pp. 22–31.
- [3] H. Kopetz, G. Bauer, The Time-Triggered Architecture, Proceedings of the IEEE 91 (1) (2003) 112 – 126.
- [4] G. Heiner, T. Thurner, Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems, in: Proc. 28th Intl. Symp. on Fault-Tolerant Computing (FTCS), IEEE Computer Society, 1998.
- [5] T. Ringler, J. Steiner, R. Belschner, B. Hedenetz, Increasing System Safety for By-Wire Applications in Vehicles by Using a Time-Triggered Architecture, in: W. Ehrenberger (Ed.), Proc. 17th Intl. Conf. on Computer Safety, Security and

Reliability (SAFECOMP), Vol. 1516 of Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 243–253.

- [6] G. Bauer, H. Kopetz, W. Steiner, Byzantine Fault Containment in TTP/C, in: Proc. Intl. Workshop on Real-Time LANs in the Internet Age (RTLIA), 2002, pp. 13–16.
- [7] G. Bauer, H. Kopetz, W. Steiner, The Central Guardian Approach to Enforce Fault Isolation in the Time-Triggered Architecture, in: Proc. 6th Intl. Symp. on Autonomous Decentralized Systems (ISADS), 2003, pp. 37–44.
- [8] H. Pfeifer, D. Schwier, F. von Henke, Formal Verification for Time-Triggered Clock Synchronization, in: C. Weinstock, J. Rushby (Eds.), Dependable Computing for Critical Applications (DCCA) 7, Vol. 12 of Dependable Computing and Fault-Tolerant Systems, IEEE Computer Society, 1999, pp. 207–226.
- [9] S. Katz, P. Lincoln, J. Rushby, Low-Overhead Time-Triggered Group Membership, in: M. Mavronicolas, P. Tsigas (Eds.), Proc. 11th Intl. Workshop on Distributed Algorithms (WDAG), Vol. 1320 of Lecture Notes in Computer Science, Springer-Verlag, 1997, pp. 155–169.
- [10] H. Pfeifer, Formal Verification of the TTP Group Membership Algorithm, in: T. Bolognesi, D. Latella (Eds.), Formal Methods for Distributed System Development – Proc. of FORTE XIII / PSTV XX, Kluwer Academic Publishers, 2000, pp. 3–18.
- [11] A. Bouajjani, A. Merceron, Parametric Verification of a Group Membership Algorithm, in: W. Damm, E.-R. Olderog (Eds.), Proc. 7th Intl. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT), Vol. 2469 of Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 311–330.
- [12] A. Merceron, M. Müllerburg, G. Pinna, Verifying a Time-Triggered Protocol in a Multi-Language Environment, in: W. Ehrenberger (Ed.), Proc. 17th Intl. Conf. on Computer Safety, Security and Reliability (SAFECOMP), Vol. 1516 of Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 185–195.
- [13] W. Steiner, J. Rushby, M. Sorea, H. Pfeifer, Model Checking a Fault-Tolerant Startup Algorithm: From Design Exploration To Exhaustive Fault Simulation, in: Proc. Intl. Conf. on Dependable Systems and Networks (DSN), IEEE Computer Society, 2004.
- [14] J. Rushby, An Overview of Formal Verification for the Time-Triggered Architecture, in: W. Damm, E.-R. Olderog (Eds.), Proc. 7th Intl. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT), Vol. 2469 of Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 83–105.
- [15] J. Rushby, Systematic Formal Verification for Fault-Tolerant Time-Triggered Algorithms, IEEE Trans. on Software Engineering 25 (5) (1999) 651–660.
- [16] S. Owre, J. Rushby, N. Shankar, D. Stringer-Calvert, PVS: An Experience Report, in: D. Hutter, W. Stephan, P. Traverso, M. Ullman (Eds.), Applied

Formal Methods (FM-Trends), Vol. 1641 of Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 338–345.

- [17] G. Bauer, H. Kopetz, P. Puschner, Assumption Coverage under Different Failure Modes in the Time-Triggered Architecture, in: Proc. 8th IEEE Intl. Conf. on Emerging Technologies and Factory Automation (ETFFA), 2001, pp. 333–341.
- [18] TTTech, Time-Triggered Protocol TTP/C High-Level Specification Document, <http://www.tttech.com/technology/specification.html> (2003).
- [19] H. Kopetz, The Time-Triggered (TT) Model of Computation, in: Proc. 19th IEEE Real-Time Systems Symposium, 1998, pp. 168–177.

On Exploiting Symmetry To Verify Distributed Protocols*

Marco Serafini and Neeraj Suri

DEEDS Group

Department of Computer Science

Technische Universitat Darmstadt (TUDA), Germany

{marco,suri}@deeds.informatik.tu-darmstadt.de

Péter Bokor

FTSRG Group and DEEDS Group (at TUDA)

Department of Measurement and Information Systems

Budapest University of Technology and Economics

petbokor@mit.bme.hu

1. Introduction

Amongst varied V&V approaches, formal verification is seeing increased usage and acceptance. One popular technique is *model checking* that allows for *automated* verification (without user guidance) by performing exhaustive simulation on the model of the system. However, model checking faces the problem of *state space explosion*. Our ongoing research aims at proposing a method to facilitate the model checking of a class of distributed protocols by reducing the size of the associated state space.

Abstraction is a general method to reduce the level of detail in order to reduce the complexity of analysis. We present an abstraction scheme which is aimed at verifying *core fault tolerant protocols of frame-based, synchronous systems* (e.g consensus, atomic broadcast, diagnosis, membership). The idea is that, instead of modeling every single node explicitly, we represent only one correct node (we term this the *one-correct-node abstraction*) that captures overall symmetry as feasible. The rationale for our abstraction is that the protocols under analysis often entail symmetry in their assumptions (A1) *synchronous* and *symmetric communication* among correct nodes, that (A2) every correct node executes the same program, (A3) a *quorum* of correct nodes, stated by the fault assumption, is always present in the system. Furthermore, we are interested in verifying (A4) *non-uniform properties*, i.e., no restrictions are required on the internal state of faulty processors.

Note that even when (A1)-(A3) hold, the internal state of correct processors may differ, e.g. due to asymmetric broadcast of a faulty node. To capture this *asymmetry* we assign *non-deterministic* values to the possibly affected variables. In spite of that, the amount of correct information symmetrically exchanged by the quorum of correct nodes must be sufficient to guarantee a coordinated behavior of the quorum itself.

*This research is supported in part by Microsoft Research, FP6 IP DE-COS and NoE ReSIST

Section 2 presents a case study to demonstrate the idea of our abstraction. For any abstraction it is necessary to prove that the abstraction function is *sound* and *complete*, i.e., it is *property preserving*. In general the abstraction is said to be property preserving if for any valid formula f^A the following holds: $M^A \models f^A \Leftrightarrow M \models f$, where M (f) and M^A (f^A) stand for the concrete and abstract model (formula) respectively. Section 3 elaborates how to prove the soundness of our abstraction scheme, while Section 4 describes our future work.

2. Interactive Consistency - A Case Study

The problem of distributing data consistently in the presence of faults is variously called interactive consistency, consensus, atomic broadcast, or Byzantine agreement. When a node transmits a value to several receivers, we mandate that the properties of *agreement* (all non-faulty receivers adopt the same value, even if the transmitting node is faulty) and *validity* (if the transmitter is non-faulty, then non-faulty receivers adopt the value actually sent) hold. In this example we consider an interactive consistency protocol consisting of many parallel instances of the classical binary consensus [3] where the maximum number of byzantine faults is $m = 1$. The protocol proceeds through of the following steps: *Step 1* every processor i acts as a sender and broadcasts its value v_i , *Step 2* every processor j receives and re-broadcasts v_{ij} to every other node, *Step 3* all nodes receive and adopt a value using a deterministic function.

Table 1 depicts the internal state of a correct node after executing the first two steps of this protocol. In this example the system contains $n = 4 > 3m$ nodes and node 3 is byzantine. In the table, v_{ij} refers to the value sent by processor j in *Step 1* as received by i and re-broadcasted in *Step 2*. The fact that nodes do not re-broadcast their own value in *Step 2* is expressed by $v_{ii} = \text{'' - ''}$. Without loss of generality we consider the case where correct nodes send 0 as their own value. Since node 3 is faulty it may send different values to the other nodes in *Step 1*. Furthermore, in

Re-broadcaster	Senders			
	1	2	3	4
1	-	0	1/0	0
2	0	-	1/0	0
3	1/0	1/0	-	1/0
4	0	0	1/0	-

Table 1. Interactive Consistency: internal state of a correct node (node 3 is faulty)

Step 2 the values received from the other nodes can be re-broadcasted arbitrary. This asymmetry is expressed by the values in $\{1, 0\}$ in column 3 and row 3 respectively. The majority voting of *Step 3* is executed across each column i to obtain the values to adopt for v_i .

Our abstraction scheme models all correct nodes using only *one correct node*. Faulty nodes are not modeled as agreement and validity are non-uniform properties. To model the presence of asymmetric faults we add non-determinism to the internal state of the one-correct-node: both column 3 and row 3 are assigned non-deterministic values, though with a different semantic. While the 3rd column is consistent for every correct node due to (A1), row 3 may differ from node to node (even if they are correct), as the faulty node can re-broadcast different values to different nodes in *Step 2*.

In general we need to show that the abstract properties hold. Intuitively, for *agreement*^A, we need show that for any arbitrary, non-deterministic value the majority voting gives the same outcome, while *validity*^A requires that if node i is correct (in our case nodes 1, 2 and 4) the value adopted by the one-correct-node is indeed that value sent by node i (0 in the example). Table 1 directly shows both properties to hold. For agreement, column 3 is consistent for every correct node, and majority voting over it will result in a consistent outcome. For validity, the fault assumption of [3] ($n > 3m$) guarantees a sufficient quorum of correct values to outvote the non-deterministic values, which become irrelevant. To prove that the abstraction is sound we have to show that *agreement*^A \Rightarrow *agreement* and *validity*^A \Rightarrow *validity* hold.

3. Symmetry Reduction

In general it might be cumbersome to prove the soundness of the abstraction for any valid formula. We aim at establishing a general abstraction scheme, which poses no restriction to the formula of interest, by proving that our approach is a case of *symmetry reduction* [2]. This would ensure that our one-correct-node abstraction is sound and complete. In fact, symmetry reduction, states that if

the state transition graph exhibits defined morphisms, then property preserving abstractions can be defined.

To assess the applicability of the abstraction two aspects have to be considered. First, we are interested in the *gain* of the approach, i.e. to which extent the abstraction reduces the state space. Initial measurements performed using the SRI-SAL model checker [4] on a diagnostic protocol [1] have shown a considerable reduction. Safety and liveness properties were checked on an Intel Xeon 3200 Ghz (5 GB RAM) machine. The experimental result exhibited a gain of nine order of magnitude in terms of BDD size (for $n = 4$ the number of visited states was $\sim 10^7$ and 10^{16} , respectively). Furthermore, the abstracted properties was checked within < 1 sec, while the non-abstracted ones needed up to 16 min.

The other aspect is to define the *class of protocols* for which the abstraction is applicable. So far we considered *synchronous communication* among nodes and we successfully applied the approach to membership and diagnosis algorithms, all of them as specific cases of the *consensus* problem. We would like to extend our approach also to *approximate agreement* protocols, used for clock synchronization, and consensus in *asynchronous* systems and hybrids. Note that, in general, while symmetry is ensured by the synchrony of the frame-based communication scheme, our abstraction, as it is currently defined, can not be directly applied to asynchronous protocols, where correct nodes can also receive different subsets of the values sent by the other correct nodes.

4. Conclusion & Future Work

For distributed protocols, we propose a symmetry based abstraction to decrease complexity of model checking by reducing the size of the state space. Our ongoing research addresses three aspects: (a) soundness of the abstraction by proving that it reduces to a special case of symmetry reduction [2]; (b) explore boundaries of applicability of the approach; (c) quantify effectiveness of the abstraction using different model checking techniques (e.g. symbolic and bounded model checking).

References

- [1] W. C.J., L. P., and S. N. Formally verified on-line diagnosis. *IEEE Transactions on Software Engineering*, Nov. 1997.
- [2] E. M. Clarke, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In *CAV '93: Proceedings of the 5th International Conference on Computer Aided Verification*, pages 450–462, London, UK, 1993. Springer-Verlag.
- [3] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3), 1982.
- [4] SAL. www.csl.sri.com/projects/sal.

Simulated Annealing Applied to Test Generation: Landscape Characterization and Stopping Criteria

Hélène WAESELYNCK, Pascale THEVENOD-FOSSE, Oifa ABDELLATIF-KADDOUR
LAAS-CNRS
7, Av du Colonel Roche
31077 Toulouse Cedex 4 – France
{Helene.Waeselynck, Pascale.Thevenod}@laas.fr

Abstract – This paper investigates a measurement approach to support the implementation of Simulated Annealing (SA) applied to test generation. SA, like other metaheuristics, is a generic technique that must be tuned to the testing problem under consideration. Finding an adequate setting of SA parameters, that will offer good performance for the target problem, is known to be difficult. Our measurement approach is intended to guide the implementation choices to be made. It builds upon advanced research on how to characterize search problems and the dynamics of metaheuristic techniques applied to them. Central to this research is the concept of landscape. Existing measures of landscape have mainly been applied to combinatorial problems considered in complexity theory. We show that some of these measures can be useful for testing problems as well. The diameter and autocorrelation are retained to study the adequacy of alternative settings of SA parameters. A new measure, the Generation Rate of Better Solutions (GRBS), is introduced to monitor convergence of the search process and implement stopping criteria. The measurement approach is experimented on various case studies, and allows us to successfully revisit a problem issued from our previous work on testing control systems.

Keywords – software testing, metaheuristic search, simulated annealing, measurement.

1 Introduction

Metaheuristic search techniques (Rayward-Smith et al., 1996) have proven useful to solve complex optimization problems. Their generality makes them capable of very wide application. For instance, they have gained attention in the field of software engineering (Clarke et al., 2003) and more specifically of software testing (McMinn, 2004).

Genetic algorithms (Holland, 1975) and simulated annealing (Kirkpatrick et al., 1983) are two examples of metaheuristic techniques used to automate the generation of test data. The underlying testing problems are as diverse as: structural testing (Jones et al., 1996) (Tracey et al. 1998a) (Pargas et al. 1999) (Michael et al., 2001) (Wegener et al., 2002), mutation testing (Adamopoulos et al., 2004), pre- and post-condition testing (Tracey et al., 1998b), exception testing (Tracey et al., 2000a),

determination of worst case execution time (Wegener et al. 1997) (Gross et al., 2000), testing high-level requirements of control systems (Schultz et al., 1995) (Abdellatif-Kaddour et al., 2003b) (Wegener and Buehler, 2004). Whatever the testing problem, it is first reformulated as an optimization problem with respect to some objective function called *fitness function* (usually for maximization problems) or *cost function* (for minimization problems). Then, a search run involves a series of test executions. Each execution allows a candidate solution (i.e. a generated test case, or test sequence) to be evaluated with respect to the objective function. Evaluation results are used to guide the generation of new candidate solutions in the run.

Metaheuristic search techniques are not ready-to-use algorithms. Rather, they define generic strategies that must be instantiated for the specific problem under consideration. A number of implementation choices have to be done, and these choices affect performance. It is generally recommended to experiment with alternative implementations of a strategy, or even with a variety of strategies. It is not unfair to say that the tuning of the search requires a great deal of effort, can indeed be quite expensive, and is not guaranteed to yield adequate choices. Section 2 of the paper provides a concrete example of this bad situation, which we faced while applying simulated annealing (SA) search to a testing problem. It reflects the difficulty of understanding the dynamics of metaheuristics.

Of course, the difficulty is not specific to the field of software testing. In the general literature of metaheuristics, there is active research to investigate relations between search problem characteristics and the performance of search techniques. The long-term aim is to gain both fundamental and practical insights into the matter. Central to this research is the concept of *landscape*. If we consider the fitness (or cost) as defining the “height” of a solution, the landscape metaphor gives rise to the visual image of peaks, valleys, and plateaus. Intuitively, the “relief” of the landscape should have a strong impact on the dynamics of exploration strategies. A number of measures have been proposed to characterize landscapes, and predict – or explain – the behavior of search techniques applied to them. They will be presented in Section 3. To date, the most advanced results have concerned combinatorial problems considered in complexity theory, like the Traveling Salesman Problem (TSP) or the Boolean satisfiability problem (SAT).

For testing problems, research is far less mature. Still, we are aware of recent work referring to landscape as an explanatory concept. The survey of (McMinn, 2004) revisits previous testing work by discussing the associated landscapes (e.g., it is shown that some fitness functions induce flat plateaus along which the search is provided no guidance). In the same spirit, (Wegener and Buehler, 2004) compare the relief of two candidate landscapes to choose the most adequate fitness function. Further developments in landscape analysis are expected to emerge in the near future, as advanced topics in

metaheuristic search get transferred to the field of software testing (or more generally to the field of software engineering: the manifesto by Harman and Jones, 2001, opens perspectives in that direction).

This paper is in the lineage of emerging work we have just mentioned. We investigate a measurement approach to the characterization of testing landscapes for Simulated Annealing. The aim is to support the implementation choices that have to be made to tune the search process. A failure story reported in Section 2 illustrates the significance of this issue. Section 3 presents a number of measures defined in the framework of combinatorial problems, and discusses their applicability to the testing problem under consideration. Two measures of landscape, the diameter and autocorrelation, are retained. They are experimented in Section 4, using both small examples and the case study of Section 2. The latter provides a challenging case for SA parameterization, since it defeated our previous attempts to find an adequate tuning. Besides landscape analysis, Section 5 considers the characterization of process runs, that is, of the trajectories followed when exploring the landscape. Ideally, we would like to detect runs that are unlikely to succeed before test resources are exhausted, e.g. detect runs trapped in flat regions. In this way, useless runs are stopped and the remaining resources are spent on exploring other trajectories. A new measure, the Generation Rate of Better Solutions (GRBS), is introduced to monitor runs and implement stopping criteria. Section 6 concludes with future directions.

2 Problem Statement

This section explains the motivation of our work. It first introduces the testing problem that yielded us to investigate the use of metaheuristic search. Then, we provide feedback from an experimentation of Simulated Annealing (SA) on a case study. Our experience exemplifies the difficulty of finding an adequate implementation of SA search.

2.1 The Testing Problem: Exploring Dangerous Scenarios at System Level

We have defined a strategy for testing high-level requirements of cyclic real-time control systems (Abdellatif-Kaddour et al., 2003a). The aim is to validate a system with respect to a safety-critical property, that is, to test the system and observe whether the property is violated or not. For the high-level properties we are interested in, violation typically results from improper interactions between a control program and its controlled environment, when physical faults affect the devices. Where formal verification is intractable, testing may be a pragmatic alternative. The test platform has to include a simulator of the environment that mimics the physical devices, the physical laws governing the controlled process, as well as the possible occurrence of faults in the devices. The test selection process should then try to favor those scenarios that will be the most “stressing” with respect to the target property. To address this problem, the proposed test strategy consists of a stepwise construction

of test scenarios. Each step explores continuations of the “dangerous” scenarios found at the previous step, using heuristic search techniques.

In order to illustrate the notion of stepwise construction, let us take the example of the boiler case study experimented in (Abdellatif-Kaddour et al., 2003a). The steam boiler problem (Abrial et al. 1996) is a well-known case study for which high-level requirements, control program code, and a software simulator of physical devices were made publicly available. The most critical requirement is the avoidance of boiler explosion, which occurs whenever the water level falls outside a safety range (i.e. it is either too low, or too high). The water level is controlled and monitored by means of four pumps, four pump controllers sensing the state of the pumps (open, closed), a water level sensor, and a steam sensor measuring the quantity of steam which comes out of the boiler. Any of these ten devices may fail, and will remain faulty until repaired by a human operator. The control program has to detect failures, and must either continue to operate in a degraded mode or shut the system down. An overview of this case study, and of our test platform, is provided in Appendix A.

We used our test strategy to search for explosive scenarios due to improper account of device failures. The tested system consisted of the control program connected to the simulator. A test sequence is then a sequence of timed commands sent to the simulator, to tell it to mimic the failure of a given device. As an example, command `Steam_fault(3)` triggers a steam sensor failure at cycle 3 of operation. Prior to any test experiment, we determined what would constitute a dangerous scenario for this case study: a dangerous scenario would put the system in a state such that the control program has an erroneous perception of its environment. This includes situations such as:

- The control program does not detect the failure of a device.
- The control program wrongly detects the failure of a device that is not faulty.
- The actual water level in the boiler is outside the estimation range calculated by the control program.

Triggering a dangerous situation is adopted as an intermediate test objective at step 1. For example, the test experiments revealed that the control program is unable to detect a steam sensor failure occurring at cycle 3 (cycle 3 corresponds to the control program leaving its initialization mode). We retained dangerous scenario `Steam_fault(3)` as a prefix for further exploration at step 2. We ended up with explosive scenario: `Steam_fault(3)`, `Pump2_fault(4)`, `WaterLevel_fault(8)`.

In (Abdellatif-Kaddour et al., 2003a), exploration at each step was performed using the simplest heuristic technique, namely random sampling. The technique was surprisingly effective, since we

found four different classes of explosive scenarios¹. However, blind random sampling is not expected to be sufficient in most cases. This motivated our interest in more sophisticated search techniques, and in particular simulated annealing.

The reasons why we chose SA rather than, say, genetic algorithms (GAs) were the following:

- GAs work with a population of solutions (i.e., a population of test sequences in our case), and have to let the population evolve during several generations. In contrast, SA works with one solution at a time. It was perceived that SA would be less demanding in terms of total number of test experiments. This was an important criterion, as applying test sequences to the complete system (including both the control program and the simulator), is quite costly in execution time.
- GAs involve two evolution mechanisms, namely mutation and crossover. The effectiveness of crossover depends on the ability to produce highly fit offspring from highly fit (and possibly dissimilar) parents. How to define meaningful crossover operators was far from intuitive for our testing problem. It was deemed easier to focus on mutation-like operators, exploring a set of “neighbors” of a candidate solution. SA is a typical example of metaheuristic based on this principle.

Several experiments with SA were performed, some of which are reported in (Abdellatif-Kaddour et al., 2003b). The experiments investigated different implementation choices for the boiler problem. We present these choices below, as well as the results we got.

2.2 *Implementation of Simulated Annealing*

Figure 1 presents the generic SA algorithm for a minimization problem with solution space S . It searches for a solution $s \in S$ that minimizes a cost function $f: S \rightarrow \mathbf{R}$. The algorithm works by randomly selecting a new solution s' in the neighborhood of current solution s , using neighborhood operator $N: S \rightarrow 2^S$. Solutions with lower cost ($\delta \leq 0$) are always accepted. Moves to inferior solutions ($\delta > 0$) may be accepted: the probability of acceptance depends on the magnitude of δ and on a control parameter t called temperature. Starting from t_0 , the temperature is gradually reduced during the search (according to cooling schedule C) so as to progressively decrease the acceptance rate of inferior

¹ These explosive scenarios are not due to bugs in the control program. They originate from the very definition of the degraded modes in the requirements (specification flaw). To the best of our knowledge, only one of the scenario classes was already identified in the literature.

solutions. The search is stopped when a pre-defined stopping condition becomes true (e.g., the maximal number of iterations is reached).

```

Select an initial solution  $s$ 
Select an initial temperature  $t = t_0 > 0$ 
Initialize the number of iterations  $i = 1$ 
LOOP while (stopping condition = false)
    Randomly select  $s' \in N(s)$ 
     $\delta = f(s') - f(s)$ 
    IF ( $\delta \leq 0$ ) THEN
         $s = s'$ 
    ELSE
        Generate random  $x$  uniformly in range  $[0,1]$ 
        IF ( $x < \exp(-\delta / t)$ ) THEN  $s = s'$  ENDIF
    ENDIF
     $i = i + 1$ 
     $t = C(t, i)$ 
END LOOP

```

Figure 1 – Simulated Annealing Algorithm

A number of implementation choices must be made to tune the generic algorithm to a particular search problem. For the boiler case study, the solution space S is a set of test sequences, where each sequence is encoded as a list of timed fault commands. The precise definition of S (time window of the sequences under consideration, required prefix...) depends on the step of the test strategy, and several spaces are possibly explored at each step. In any case, the aim is to find explosive or dangerous scenarios in the target S . The implementation of SA search then involves the following settings.

- Cost function f – For some testing problems, the cost function definition may be obvious. For example, WCET problems are associated with measures of the execution time. In our case, determining an adequate cost function is far from trivial. The cost function has to assign the lowest cost to test sequences reaching the test objective, yielding the general form: *IF (boiler explosion OR dangerous situation observed) THEN return (0) ELSE ...* Now, the difficulty lies in the determination of the *ELSE* part. How should the cost of two candidate sequences yielding neither an explosion nor a dangerous situation be compared? We experimented with several cost functions. Appendix B gives two examples: Cost1 was used in unpublished experiments while Cost2 is the one used in (Abdellatif-Kaddour et al., 2003b). Both functions involve some constant parameters K_i , whose value was determined empirically.

- Neighborhood N – Given a test sequence, it seems reasonable to consider neighbors that differ in only one fault command. The neighborhood operator either removes one fault, or adds one, or changes the date of occurrence (see Appendix B).
- Cooling Schedule C – Two commonly used schedules are (1) the geometric variation of temperature: $t_{i+1} = \alpha t_i$ where α is chosen close to 1.0, and (2) the schedule proposed by (Lundy and Mees, 1986): $t_{i+1} = t_i / (1 + \beta t_i)$ where β is chosen close to zero. We experimented with both. Note that calibration of parameters α (resp. β) and t_0 required preliminary executions of SA on the boiler system.
- Stopping Condition – The search is stopped when either the test objective is fulfilled ($f(s) = 0$) or the maximal number of iterations is reached ($i = MAX_ITER$). Since any execution of the boiler system is costly in time, and since the test strategy involves exploration of several search spaces, we cannot afford large values of MAX_ITER . We experimented with $MAX_ITER = 100$ and 200 . These are small values compared to common usage of SA, which gives more time for the algorithm to converge toward an optimum.

Our experience is that the implementation of SA search requires substantial effort. The tuning procedure is largely ad hoc. Many trials are necessary to investigate implementation choices and to study alternative combinations of parameter settings.

2.3 *Experimental results*

In spite of the invested effort, SA performance turned out to be quite disappointing for the Boiler example. It never outperformed random sampling. For some of the explored spaces, the high density of zero cost solutions could explain this result: any sophisticated technique was bound to be less cost-effective than random sampling. But poor results were also obtained in less trivial cases.

Let us take the example of one of the search spaces explored at step 2 of the strategy. Table 1 gives comparative results of random sampling and SA. The experiments involved 35 runs of each technique. A run is successful as soon as a zero cost sequence is found. It is stopped at the corresponding iteration. Unsuccessful runs are stopped at $MAX_ITER = 200$. Table 1 provides both the number of successful runs, and the total number of iterations for the 35 runs. Note that the two implementations of SA only differ in the cost function: all the other parameters are set the same.

	Random sampling	SA Cost1	SA Cost2
Successful search	26	25	16
Iterations (35 runs)	4187	3778	4453

Table 1 – Experimental results for one search space of the Boiler example

The best SA implementation (the one with Cost1) does not have a greater number of successful runs than random sampling. By comparing the total number of iterations, it might be concluded that SA speed is slightly higher (it takes 3778 iterations to find 25 zero cost sequences, versus 4187 iterations and 26 sequences). But the “improvement” is too insignificant to be worth the effort.

In (Abdellatif-Kaddour et al., 2003b), such results led us to propose a variant of the SA algorithm (i.e., the core algorithm in Figure 1 was modified), which performed well on the Boiler example. But we are aware that this specific variant might be inadequate for other examples. Each time a new control system is studied, the tuning effort needs to be made again and again.

This motivates our interest in investigating improvements of the tuning procedure. Indeed, for the testing problems we address, SA search cannot be a realistic option unless systematic approaches are proposed to support implementation choices. For example, the experiments of Table 1 *a posteriori* show that Cost2 is a worse choice than Cost1. Intuition is a poor guide to predict this result. There is a need for more reliable means to assess the quality of alternative settings, and this is all the more crucial as the testing time has to be kept reasonable.

It turns out that fundamental work on search techniques has settled a framework to address this issue. A number of measures have been proposed to characterize search problems and the behavior of heuristics applied to them. In this paper, we investigate the predictive power of such measures for tuning SA search in the case of testing problems.

3 Measures for Characterizing Search Problems and Metaheuristics

Fundamental research on metaheuristics aims to establish relations between measures of problems and the dynamics of metaheuristic search. The studied problems are most often NP complete problems considered in complexity theory, like graph problems or constraint satisfaction problems.

Existing measures can be classified into three broad categories, corresponding to the structure they characterize (Belaidouni and Hao, 2000). The underlying structures are:

- the *search space*, defined by a couple (S, f) where S is a solution space and f a cost function assigning a real number to each point in the space;

- the *search landscape*, defined by a triplet (S, f, N) where N is a neighborhood operator connecting each point in S to a non-empty set of neighbors;
- the *process landscape*, defined by a quadruplet (S, f, N, φ) where φ is a search process based on the notion of neighborhood.

Examples of measures for each structure are provided in Sections 3.1 to 3.3. A more complete description of measures and of related work can be found in (Belaidouni, 2001). Section 3.4 discusses applicability to our testing problem.

3.1 Characterizing the Search Space

For the search space structure, typical examples of measures are the variation range $[f_{min}, f_{max}]$ of cost values, or the number of optimal solutions in S . A finer measure concerns the *density of states* (dos), giving the frequency of each cost value in the search space (Rosé et al., 1996). For some problems, dos may be determined analytically. But most often it has to be approximated by using a sampling procedure, which can be expensive to properly account for the least frequent values.

3.2 Characterizing the Search Landscape

The search landscape has been the most studied structure. The introduction of a neighborhood operator N makes it possible to define the distance between points s and s' in S . It corresponds to the minimal number of times N must be applied to move from s to s' . The notion of distance is underlying all measures for characterizing the search landscape.

3.2.1 Diameter

The diameter D of a search landscape is the maximal distance between two points in S . Intuitively, small diameters should be more favorable than large ones, because any point can potentially be reached quicker (in terms of successive applications of N). Suppose D is large. Then, it may take a large number of iterations to get to an optimal solution, even in the ideal case where the search process would select the shortest trajectory between the starting point and the target solution.

The diameter can usually be determined analytically (for landscapes of typical NP-complete problems, see examples of analytical expressions in Angel and Zissimopoulos, 2000).

3.2.2 Autocorrelation

The autocorrelation measure has been introduced by (Weinberger, 1990) to characterize the *ruggedness* of a landscape. Intuitively, a smooth landscape is one in which neighbors have nearly the

same cost, while a rugged one is one in which the cost values are dissimilar. The latter situation indicates that f and N are not mutually adequate to guide the search.

The autocorrelation ρ_d measures the variation of cost for points that are at distance d . Most often, measurement is focused on ρ_1 , which is deemed the most important value to know. Exact evaluation of the autocorrelation is not possible unless the distribution of cost values is known. In practice, ρ_1 is estimated by means of a random walk whose steps consist in moving to a new point chosen randomly among the neighbors of the current point (see Figure 2). The underlying assumption is that the statistics of the cost sequences generated by a random walk are the same, regardless of the starting point chosen: the landscape has to be *statistically isotropic* (Weinberger, 1990). An estimate of ρ_1 is then (Hordijk, 1996):

$$\hat{\rho}_1 = \frac{\sum_{i=0}^{T-2} (f(s_i) - \bar{f})(f(s_{i+1}) - \bar{f})}{\sum_{i=0}^{T-1} (f(s_i) - \bar{f})^2}$$

where T is the size of the sample and \bar{f} the arithmetic mean of the cost values in the sample. If ρ_1 is close to one, the landscape is smooth. If ρ_1 is close to zero, neighboring cost values are unrelated and neighborhood search techniques are not expected to be effective.

Autocorrelation (or derived measures) has been used to explain search performance on landscapes associated with NP-complete problems (Stadler and Schnabl, 1992) (Angel and Zissimopoulos, 1998) (Angel and Zissimopoulos, 2000) as well as on artificial landscapes whose ruggedness can be tuned by control parameters (Hordijk, 1996).

```

Select a starting point  $s_0$ 
Sample[0] =  $f(s_0)$ 
FOR  $i=1$  to  $T-1$  DO
    Randomly select  $s_i \in N(s_{i-1})$ 
    Sample[ $i$ ] =  $f(s_i)$ 
END FOR

```

Figure 2 – Random Walk Algorithm

3.2.3 Other Measures

While autocorrelation characterizes the variation of costs at a fixed distance, other measures are focused on the variation of distances at a fixed cost. For example, such measures are used by (Belaidouni and Hao, 2000) to study landscapes of the MAX-CSP problem. A practical difficulty is

that their experimental evaluation requires the production of a sample of solutions for each cost level. In (Belaidouni and Hao, 2000), the authors have to apply metaheuristic search to produce the samples. There are also measures characterizing the joint variation of distances and costs, like the Fitness Distance Correlation (FDC) (Jones and Forrest, 1995), used in the framework of genetic algorithms and genetic programming (Vanneschi et al., 2003). The FDC definition involves the notion of distance to the nearest global optimum, which restricts its applicability to problems with known optima.

Finally, it is worth mentioning work studying the topology of landscapes in terms of local optima and plateaus, which are regions the search can be trapped in (Frank et al., 1997). For small instances of problems, the number and size of such regions can be exhaustively determined, as in (Yokoo, 1997). Also, for some simple landscapes, the number of local optima can be approximated analytically (see e.g. the TSP landscape analysis by Stadler and Schnabl, 1992). However, in the general case, approximation has to be achieved via expensive experiments (Eremeev and Reeves, 2003).

3.3 Characterizing the Process Landscape

The process landscape should be the most informative structure, since it accounts for all parameters of the search implementation. It is also the most difficult (and the most expensive) to characterize, because analytical analysis is not possible and empirical analysis involves running the search process.

Most of the measures defined for the search space and search landscape can be transferred to the process landscape. For example, the process cost density (pcd) (Belaidouni and Hao, 2002) is analogous to the density of states (dos) defined for search spaces (cf. Section 3.1), using process φ as the sampling procedure. It is worth noting that dos and pcd are not the same, because φ introduces a bias in the exploration of solutions. Similarly, the number and size of basins of attractions for φ should be related to the local optima of the search landscape, but the relation is not trivial.

When characterizing the process landscape, an acute problem is the variability of behavior from one run of φ to the next: obtaining statistically meaningful measures can be practically impossible. The problem already existed for the characterization of search landscapes using a random walk. However, it is much more acute when the samples are produced by sophisticated metaheuristics. Indeed, for the boiler example, we empirically observed variability of SA behavior, depending on the starting point of the search.

3.4 Discussion

While a number of measures have been proposed in the literature, there is currently no consensus on which ones to use in which case. Generally speaking, a single measure can only have a limited

explanatory power. For example, autocorrelation is one of the most acknowledged measures but can be insufficient to distinguish between two search landscapes of different difficulty.

In spite of these limitations, we will show that a measurement approach can still be useful to implement SA search for our testing problem. The choice of measures has to be guided by our specific needs. First, we cannot afford a large number of experiments to get an estimation of measures, because execution of the complete system is computationally costly. Second, our aim is to tune SA search to get zero-cost solutions, not just low cost solutions. This is so because non-zero solutions do not fulfill the test objective: in the boiler example, they correspond to safe scenarios triggering neither a boiler explosion nor a dangerous situation. Let us discuss the implication of these needs.

The constraint in the number of system executions leads us to exclude the most costly measures for characterizing the search landscape (e.g., the measures mentioned in Section 3.2.3), as well as all measures for characterizing the process landscape (see Section 3.3). As regards the latter measures, let us recall that their estimation is anyway problematic due to the variability of SA behavior.

The fact that we are not interested in sub-optimal solutions limits the insight that can be gained from some measures. Let us take the example of *dos* and *pcd*. A search space will be considered all the easier as the cost distribution exhibits a low mean and high variance (i.e., it is not difficult to find low cost solutions, and very low values departing from the mean are not too scarce). Similarly, a process landscape will be considered better than another if *pcd* analysis shows that it tends to generate lower cost solutions. Such results are relevant to problems for which low cost, but possibly sub-optimal, solutions are quite acceptable. They are not relevant in our case because sub-optimal solutions are useless with respect to the test objective, as previously explained.

We finally retain two measures for characterizing the search landscape: the diameter D and autocorrelation ρ_l . They will be used in Section 4 to guide the choice of “good” combinations of a cost function f and neighborhood operator N , for SA search.

How to characterize the process landscape remains an open issue. In Section 5, we will adopt a less ambitious approach. We will not attempt to refine tuning of the process landscape. Rather, we will attempt to make the best possible use of a given process landscape. Since behavior depends on the starting point of the search, we will try to detect runs that are unlikely to succeed in the allowed number of iterations, so as to stop them and restart search from another point. This will lead us to propose a new measure to monitor the behavior of a run and implement stopping criteria.

4 Characterizing the Search Landscape

The diameter and autocorrelation measures are used to identify search landscapes that are potentially well-suited to SA search. The tuning process shall retain settings of f and N such that:

- Diameter D is significantly lower than MAX_ITER , the allowed number of iterations per run.
- Autocorrelation ρ_l is high.

The diameter criterion is intended to ensure that an optimal solution has a chance to be reached before the end of the run, whatever the starting point of the search. The autocorrelation criterion identifies landscapes for which the principle of neighborhood search is not irrelevant. Equipped with these criteria, our challenge is to revisit the boiler example and to determine an adequate parameterization of SA search for it.

Before addressing the boiler example, we study the relevance of the criteria for simpler problems. The first one, QAP (Section 4.1), has been extensively studied in the literature of metaheuristic search. It is considered here for comparison purposes. It provides a typical example of the combinatorial problems to which diameter and autocorrelation measures are usually applied. It is also one of the problems for which SA search is known to be efficient. The next two problems, Cal1 and Cal 2 (Section 4.2), are loosely inspired from a calendar testing problem. They are used to perform a large number of controlled experiments on “good” and “bad” landscapes. We let f and N vary, and try to establish a relation between the corresponding D and ρ_l measures on the one hand, and SA efficiency on the other hand. This makes it possible to study the discriminative power of the criteria. The boiler example is then revisited in Section 4.3.

4.1 The QAP Landscape

The Quadratic Assignment Problem (QAP) is a typical representative of combinatorial problems studied in the literature of metaheuristic search. It is a NP-hard problem generalizing the Traveling Salesman Problem. A number of search techniques have been experimented on QAP instances (Connolly, 1990) (Taillard, 1991) (Maniezzo et al., 1995) (Merz and Freisleben, 2000). SA search is known to be very efficient for QAP, using some standard settings of f and N . Actually, cost function f is already determined by the problem definition, and it is only N that needs to be adequately chosen. The standard N for QAP is the so-called *2-exchange neighborhood* (see e.g., Angel and Zissimopoulos, 2000, for a definition of this neighborhood operator). Thus, the QAP example provides us with the opportunity to investigate characterization of a landscape known to be “good”.

The experiments involve the NUG15 instance of QAP proposed by (Nugent et al., 1968)². It has four solutions supplying an optimal cost, in a search space of $15!$ elements. In order to check our ability to reproduce known results, we retain the same SA implementations as the ones described in (Connolly, 1990). The maximal number of iterations is taken as $MAX_ITER = 5000$ to keep the same order of magnitude as in Connolly’s experiments (which involved 5250 iterations for this problem instance). Before experimenting with SA search, we first characterize the underlying search landscape.

Diameter criterion – With the 2-exchange neighborhood, $D = 14$ which is obviously much lower than the allowed number of iterations.

Autocorrelation criterion – The experimental evaluation of ρ_l involved 100 random walk runs (see Figure 2 for the random walk algorithm), with 1000 iterations per run. Note that this effort is unusually high as a single run should be necessary to perform estimation: we wanted to confirm the expected stability of estimates for this landscape. We obtained $\rho_l = 0.74$ (standard deviation 0.01). This gives an idea of what should be considered a sufficiently high autocorrelation measure.

For this “good” landscape, we confirm the expected efficiency of SA search. In Table 2, L&M1 and L&M2 correspond to the two SA implementations described in (Connolly, 1990): L&M1 is standard SA with the Lundy and Mees cooling schedule, and L&M2 is a variant that yielded better results for a range of QAP instances. Whatever the SA implementation, any of the four optimal solutions is found by at least 21% of successful runs. Whereas random sampling fails to produce any optimal solution, SA search is efficiently guided by the structure of the underlying landscape.

	Random sampling	L&M1	L&M2
Percentage of successful runs (1000 runs, $MAX_ITER = 5000$)	0%	8%	11.1%

Table 2 – SA efficiency for QAP (NUG15 instance)

4.2 Landscapes for two Calendar Problems

After having characterized a known landscape, we now focus on landscapes for two artificial problems, loosely inspired from a calendar problem experimented by (Tracey, 2000b) in the field of software testing. For these two artificial problems, noted Cal 1 and Cal 2, the solution space is the set of dates from January 1, 1900 up to December 31, 3000. Each date is encoded by a triplet (*day*, *month*,

² This problem instance, as well as many other ones, is archived on Eric Taillard’s page: <http://ina.eivd.ch/Collaborateurs/etd/problemes.dir/qap.dir/qap.html>

year). We make the simplifying assumption that there are exactly 365 days per year (there is no account for leap years), hence the space contains 401,865 solutions. The search problem is then to find a zero cost date in the space. Cal 1 has exactly one optimum to be found (arbitrarily taken as December 31, 2000), while Cal 2 has two (December 31, 2000 and February 2, 2500). In each case, we experimented with 24 different search landscapes, by making f and N vary. Cal 1 is used to illustrate the results below. A complete report of all experiments can be found in (Abdellatif-Kaddour, 2003c).

The four cost functions and six neighborhood operators used for Cal 1 are described in Appendix C. Basically, *Cost1* is a random cost function that is expected to yield poor results whatever the neighborhood definition. *Cost2*, as well as neighborhood operators *N1* and *N2*, are inspired from Tracey's work, and should fit well together. *Cost3* and *Cost4* have been designed in the same spirit as *N3-N6*, so that any combination of them is expected to supply a high correlation of neighboring costs. The Cal 2 cost functions are similar to the Cal 1 ones, but are slightly adapted to account for the additional optimum. The neighborhood operators are kept exactly the same.

Let us now characterize the landscapes resulting from all $f \times N$ combinations. The maximal number of iterations per run is set to 2000.

Diameter criterion – Table 3 shows the diameter values for all neighborhood operators. Bold values indicate that the criterion is passed, while italic values denote diameters that are judged too large compared to the allowed number of iterations. As can be seen, *N3* and *N4* are rejected as bad choices for $MAX_ITER = 2000$.

Autocorrelation criterion – Table 4 shows ρ_l estimates for the 24 landscapes of Cal 1. In each case, experimental evaluation involved 100 random walk runs, with 1000 iterations per run. A cell of the table indicates the mean value of ρ_l , followed by the standard deviation put in brackets. Ten landscapes (with ρ_l in italic numbers) do not pass the autocorrelation criterion. Not surprisingly, all landscapes involving *Cost1* are rejected. Moreover, we learn that *N3-N6* are not well-suited to *Cost2*, while *N1-N2* fit well with *Cost3-Cost4* although they have not been designed for that purpose. Cal 2 landscape analysis yields similar conclusions.

Putting together the diameter and autocorrelation criteria, 10 landscapes are retained for each of the Cal 1, Cal 2 problems, and 14 landscapes are rejected. We then assess the relevance of these decisions by measuring SA search efficiency for all landscapes. Table 5 gives the percentage of successful runs for the Cal 1 problem, observed from a sample of 1000 SA runs per landscape. It can be seen that the 14 rejected landscapes (whose results are displayed in italic characters) exactly correspond to the 14

worst scores. This result is encouraging, in spite of the fact that the diameter and autocorrelation measures have a limited discriminating power: the 10 retained landscapes do supply the best scores but these scores exhibit a large variation (ranging from 6.3% to 65.1%). Note, however, that the lowest score (6.3%) is still significantly higher than that of random sampling, which was 0.4% in the Cal 1 case study. Similar results are obtained for Cal 2.

Hence, despite limitations, the proposed approach should be relevant to reject bad landscapes, and retain only those that are reasonably well suited to SA search (while possibly not “best” suited). We are now equipped to revisit the Boiler example.

	N1	N2	N3	N4	N5	N6
Diameter	550	1100	<i>6700</i>	<i>13400</i>	250	500

Table 3 – Diameter of the calendar landscapes

	N1	N2	N3	N4	N5	N6
Cost1	<i>0 (0)</i>					
Cost2	0.82 (0.08)	0.95 (0.04)	<i>0.22 (0.09)</i>	<i>0.22 (0.09)</i>	<i>0.36 (0.19)</i>	<i>0.33 (0.17)</i>
Cost3	0.99 (<10⁻²)	0.99 (<10⁻²)	0.89 (<10⁻²)	0.89 (<10⁻²)	0.98 (0.02)	0.98 (0.01)
Cost4	0.99 (<10⁻²)					

Table 4 – Autocorrelation of the Cal 1 landscapes: mean (stand. dev.)

	N1	N2	N3	N4	N5	N6
Cost1	<i>0%</i>	<i>0%</i>	<i>0%</i>	<i>0%</i>	<i>0.2%</i>	<i>0.2%</i>
Cost2	6.3%	16.1%	<i>0%</i>	<i>0%</i>	<i>1.4%</i>	<i>1.4%</i>
Cost3	47.6%	65.1%	<i>2.1%</i>	<i>2.1%</i>	16.9%	16.8%
Cost4	60.8%	9.5%	<i>3.7%</i>	<i>3.7%</i>	40.3%	40.2%

**Table 5 – Percentage of successful SA runs on the Cal 1 landscapes
(1000 runs, MAX_ITER=2000)**

4.3 The Boiler Search Landscape

Compared to the previous case studies, the Boiler example is one for which random sampling is fairly efficient. The most difficult search space is the one mentioned in Section 2.3, where random sampling found 26 zero-cost solutions using a total number of 4187 iterations (see Table 1). We will focus on the corresponding search problem. As regards SA efficiency, we observed that the choice of the cost function had a strong impact, but the best function proposed so far did not significantly outperform random sampling. We will try to improve the situation by tuning the cost function. The size of the target search space is 10,077,695. The allowed number of iterations is $MAX_ITER = 200$.

Let us now characterize the two landscapes studied in Section 2.3.

Diameter criterion – The chosen neighborhood operator yields $D = 9$, hence the diameter criterion is passed.

Autocorrelation criterion – For both landscapes, the experimental evaluation of ρ_l involved 10 random walk runs, with 100 iterations per run. Table 6 shows the results. The autocorrelation measure does not discriminate between Cost1 and Cost2. In both cases, neighboring costs are not unrelated but ρ_l values are smaller than the ones accepted for the QAP, Cal 1 and Cal 2 problems (the retained landscapes always had $\rho_l \geq 0.74$). Hence, the results do not reveal strong inadequacy of the landscapes, but suggest that it may be preferable to design a cost function with higher autocorrelation.

	Mean	Stand. Dev.
Cost1	0.52	0.17
Cost2	0.53	0.13

Table 6 – Autocorrelation of the Boiler landscapes

Rather than inventing an entirely new cost function, we go back to the generic definitions of Cost1 and Cost2 in Appendix B, expressed in terms of symbolic parameters K_i . Let us recall that the K_i calibration was based on trials and empirical judgment. We now try a more systematic approach, by making the K_i values vary, and retain the valuation supplying the best autocorrelation.

As regards Cost1, we observed the best autocorrelation for $K_1 = 100$, $K_2 = 0$, $K_3 = 10000$. Improvement is not drastic, since $\rho_l = 0.62$ (stand. dev. 0.14). Still, we retain the corresponding cost function, noted $Cost1_{bis}$.

As regards Cost2, we failed to obtain any significant improvement. This may suggest that the generic definition is not well suited to the search problem and/or the chosen neighborhood operator.

We now study whether the new Cost 1_{bis} function manages to improve SA efficiency. Table 7 provides the results obtained by 35 runs ($MAX_ITER = 200$). The previous results of random sampling and Simulated Annealing are recalled for comparison purposes. It is clear that the Cost 1_{bis} landscape is the best for SA search. Moreover, observed performance is now significantly better than that of random sampling. Considering our initial failure to properly tune SA search, these results confirm that the proposed measurement criteria should be much sounder than *ad hoc* judgment.

	Random sampling	SA Cost1	SA Cost2	SA Cost1_{bis}
Successful search	26	25	16	33
Iterations (35 runs)	4187	3778	4453	2573

Table 7 – SA search improvement for the Boiler example

5 Stopping Criteria for Process Runs

Contrary to what has just been done on the search landscape, we do not try to improve the process landscape. For the reasons explained in Section 3.4, our objective is more modest: we attempt to make the best possible use of a given process landscape. This is done by monitoring the process runs. Each run corresponds to a trajectory in the landscape, which may or may not reach an optimal solution depending on the starting point of the search and the allowed number of iterations. The monitoring approach aims to decide whether the current run is likely to succeed before the allowed number of iterations. If the decision is negative, the run is stopped and search is restarted from another point.

Stopping criteria have also been studied by others in the framework of search-based testing. In (Lammermann and Wegener, 2005), the allowed search effort is tuned to structural coverage goals: the stopping criteria are *a priori* determined from software measures. In (O’Sullivan et al., 1998), on-line monitoring of search convergence is used to determine whether the current run should be stopped or continued. As an *a priori* estimation is impossible for us, our approach is more in the spirit of the latter work. However, its implementation is quite different since (O’Sullivan et al., 1998) use genetic algorithms and analyze convergence for populations of solutions. Section 5.1 presents the principle of our monitoring approach and the related stopping criteria. Section 5.2 uses the QAP and calendar problems to calibrate the criteria. A trade-off must be found between too sensitive criteria (stopping runs that would have been successful) and too restrictive ones (not stopping unsuccessful runs). The calibrated criteria are then applied to the Boiler example in Section 5.3.

5.1 Principle of the Approach

A new measure is introduced to characterize a process run, called the Generation Rate of Better Solutions (GRBS):

$$\text{GRBS} = \frac{\text{number of generated solutions decreasing the cost}}{\text{total number of generated solutions}}$$

The GRBS value is not very interesting *per se* because it is unstable, evolves during a run, and varies from one run to the next. Actually, it is precisely the evolution during a run that is the focus of our monitoring approach.

Whatever the case study (QAP, calendar problems, Boiler), we observed that GRBS evolution for *successful* runs can be schematized as in Figure 3. Three phases may be identified. Phase 1 typically characterizes the beginning of the search at a point where cost is high and easy to improve: lower cost solutions are produced at a high rate. At the end of Phase 1, low cost solutions are reached and improvement becomes difficult: the steep decrease of GRBS (Phase 2) indicates that search is approaching the optimal solution that will be found by this successful run. Phase 3 characterizes the last iterations of the run, exploring a small subspace in the neighborhood of the optimal solution. Of course, Figure 3 is only a schematic view. The actual duration of each phase depends on the run, and it may be the case that a phase is not present. Concrete examples of runs with no Phase 1 or no Phase 3 are shown in Figure 4.a.

An interesting observation is that the duration of phases can – to a certain extent – be used to distinguish successful and unsuccessful runs. We observed empirically that *unsuccessful* runs always correspond to long durations of either Phase 1 or Phase 3, where “long” refers to the maximal number of iterations allowed for the runs. Concrete examples are shown in Figure 4.b.

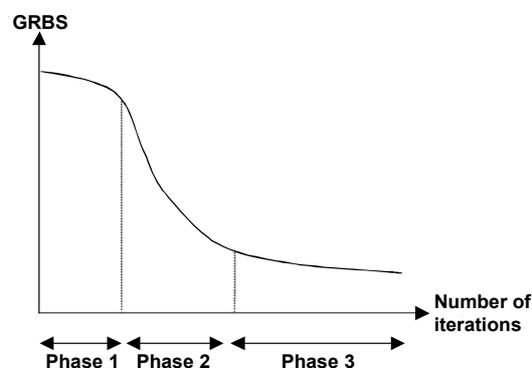
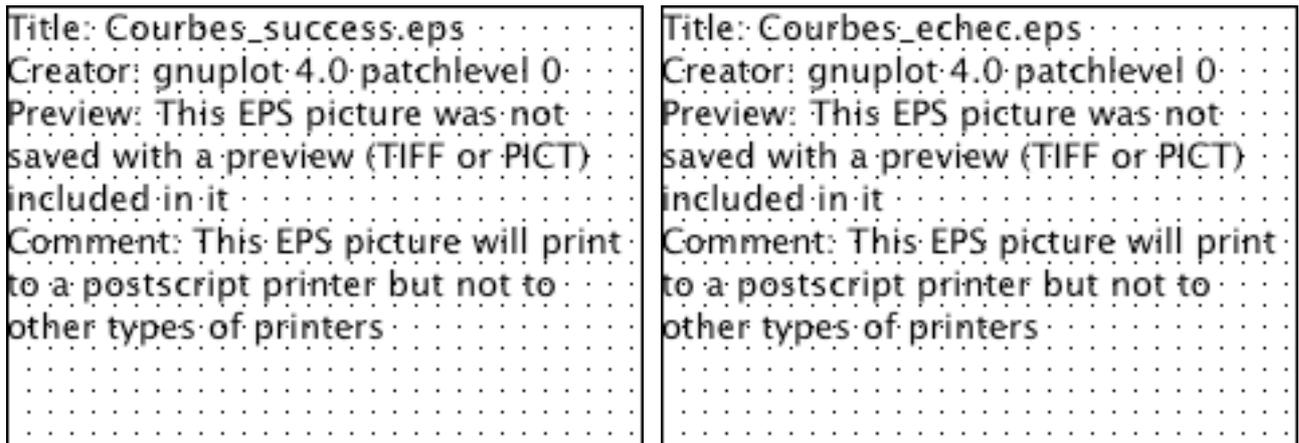


Figure 3 – Generic GRBS evolution for successful runs



(a) Examples of successful runs

(b) Examples of unsuccessful runs

Figure 4 – Instances of GRBS evolution (Cal 1, Cost 4 x N6 landscape)

A long Phase 1 yields high GRBS values to be collected during the whole run. The starting point of the search has a high, easy to improve cost. Still, it turns out that the process moves fail to reach a low cost configuration: better solutions are continuously produced at a high rate, but each solution only provides a slight improvement compared to the previous one. Hence, when the maximum number of iterations is reached, the achieved cost value is still far away from the optimum.

For unsuccessful runs with a long Phase 3, it is observed that after transient phases 1 and 2, better solutions are scarcely produced during a large number of iterations (until MAX_ITER is eventually reached). The stagnation may indicate that the search process is stuck at a local optimum or trapped in a flat region.

It would be desirable to stop such runs as soon as possible, to avoid useless iterations. Two complementary stopping criteria are proposed, based on the on-line monitoring of the GRBS. The first (resp. second) criterion determines whether Phase 1 (resp. Phase 3) is too long for the run to possibly succeed within the allowed number of iterations. Both checks are performed in parallel, and the run is stopped whenever one criterion becomes true.

- **Phase 1 too long – $GRBS \geq Threshold_High$ during $max(\alpha MAX_ITER, D)$ iterations.**

In the proposed formulation, Phase 1 of the search is identified by a GRBS value exceeding a threshold high value. The judgment that Phase 1 is “too long” must then be relative to the maximal number of iterations: this is tuned by means of parameter α with $0 < \alpha < 1$. In addition to this, it seems reasonable not to stop the run before the process has spent some minimal amount of time to explore the landscape. In the proposed formulation, the exploration time cannot be shorter than diameter D of the landscape.

- **Phase 3 too long – $GRBS \leq Threshold_Low$ during $max(\beta MAX_ITER, |N|)$ iterations.**

As above, the phase duration is tuned to the maximal number of iterations: this is done by means of parameter β with $0 < \beta < 1$. In addition to this, we impose that a Phase 3 search cannot be stopped before it has spent some time to explore the neighborhood of the solution reached at the end of Phase 2. This is why the triggering number of iterations cannot be smaller than the size $|N|$ of the neighborhood operator, where $|N|$ is defined as the maximal number of neighbors connected to any point.

In order to implement the checks, parameters α , β , $Threshold_High$ and $Threshold_Low$ must be instantiated. We experimented with a number of values using the QAP, Cal 1 and Cal 2 case studies. We also tried several monitoring schemes: on-line monitoring is started at the first iteration of the run, or after an initial number of iterations has been performed.

5.2 Calibration using QAP, Cal 1 and Cal 2

The best monitoring schemes, that consistently supplied satisfactory results for all case studies, were the following:

Scheme 1: Monitoring is started after $MAX_ITER/10$ iterations.
Phase 1 too long: $Threshold_High = 50\%$, $\alpha = 0.1$
Phase 3 too long: $Threshold_Low = 10\%$, $\beta = 0.1$

Scheme 2: Monitoring is started at the first iteration.
Phase 1 too long: $Threshold_High = 50\%$, $\alpha = 0.25$
Phase 3 too long: $Threshold_Low = 10\%$, $\beta = 0.25$

It is worth noting that according to Scheme 1 (resp. 2), the search can never be stopped before $MAX_ITER/5$ (resp. $MAX_ITER/4$) iterations.

The experiments involved the two process landscapes of QAP, and the 20 landscapes of Cal 1 and Cal 2 that passed the diameter and autocorrelation criteria (Abdellatif-Kaddour, 2003c). Exemplary results for the two retained schemes are shown in Table 8. They have been obtained for the QAP landscapes, by re-playing the runs from Section 4.1 with the monitoring schemes. The replay allows determination of the percentage of unsuccessful – but also successful – runs that are stopped by each scheme. Global search performance is then measured by the ratio (number of successes) / (total number of iterations). It accounts for the combined effect of false positives (successful runs are stopped) and false negatives (unsuccessful runs are not stopped).

	L&M1			L&M2		
	Without monitoring	Monitoring scheme 1	Monitoring scheme 2	Without monitoring	Monitoring scheme 1	Monitoring scheme 2
% unsucc. runs stopped	—	100%	100%	—	100%	100%
% succ. runs stopped	—	76.25%	50%	—	60.36%	18.92%
Success/iteration	$1.7 \cdot 10^{-5}$	$1.9 \cdot 10^{-5}$	$2.4 \cdot 10^{-5}$	$2.4 \cdot 10^{-5}$	$4.3 \cdot 10^{-5}$	$5.5 \cdot 10^{-5}$

Table 8 – Monitoring schemes applied to the QAP landscapes

From Table 8, it can be seen that all unsuccessful runs are detected and stopped. Unfortunately, the monitoring schemes may also stop a high percentage of *successful* runs. But we experimentally observed that those runs are most often the ones that succeed lately, i.e. towards the end of the 5000 allowed iterations. This explains why global search performance is not degraded.

For the two QAP landscapes, Scheme 2 turns out to be the best one. The supplied improvement is the same order of magnitude as the one supplied by using Connolly’s L&M2 – rather than L&M1 – search process. However, considering the whole set of 22 landscapes, no scheme exhibits superiority over the other. The conclusions drawn from the 22 landscapes are the following:

- Both schemes manage to stop a satisfactory percentage of unsuccessful runs (> 50% for 12 landscapes).
- Successful runs are also stopped (< 20% for 13 landscapes), but these are generally runs that would have consumed almost *MAX_ITER* iterations before an optimal solution is found.
- Global search performance is either improved or unchanged, never degraded.

Hence, it seems that the monitoring schemes can significantly improve performance in some cases (as for the QAP example), and have a neutral effect in the worst case. It remains to be studied whether similar results can be obtained for the Boiler problem.

5.3 Experimentation with the Boiler Example

The retained schemes have been experimented on three Boiler landscapes, namely the ones with Cost1, Cost2 and Cost1_{bis}. The results shown in Table 9 correspond to the same runs as the ones in Section 4.3, replayed with monitoring.

	Cost1			Cost2		
	Without monitoring	Monitoring scheme 1	Monitoring scheme 2	Without monitoring	Monitoring scheme 1	Monitoring scheme 2
# unsucc. runs stopped	—	10 (out of 10)	10 (out of 10)	—	5 (out of 19)	5 (out of 19)
# succ. runs stopped	—	11 (out of 25)	12 (out of 25)	—	0 (out of 16)	0 (out of 16)
Success/iteration	$6.6 \cdot 10^{-3}$	$7.2 \cdot 10^{-3}$	$8.2 \cdot 10^{-3}$	$3.6 \cdot 10^{-3}$	$4.0 \cdot 10^{-3}$	$4.0 \cdot 10^{-3}$

(b) Cost1 and Cost2 landscapes

	Cost1 _{bis}		
	Without monitoring	Monitoring scheme 1	Monitoring scheme 2
# unsucc. runs stopped	—	1 (out of 2)	1 (out of 2)
# succ. runs stopped	—	2 (out of 33)	2 (out of 33)
Success/iterations	$12.8 \cdot 10^{-3}$	$13.0 \cdot 10^{-3}$	$13.3 \cdot 10^{-3}$

(a) Cost1_{bis} landscape

Table 9 – Monitoring schemes applied to the Boiler landscapes

For these landscapes, search performance is similar whether or not the monitoring schemes are used. Note that, for the Cost1_{bis} landscape, no drastic improvement could be expected because SA search is already quite efficient (indeed, there are only 2 unsuccessful runs). At least, our results confirm that performance is not degraded, which is also the case for Cost1 and Cost2 landscapes. This outcome was not granted, because the allowed number of iterations (200) is much lower than the one experimented for QAP (5000) or the calendar problems (2000). For such comparatively short runs, we could not be sure that the monitoring schemes would still be relevant. The fact that behavior remains consistent with previous observations is a positive point in favor of the reusability of the retained schemes. As previously, the stopped successful runs are the ones that succeed lately.

Our conclusions are then the following. Since the monitoring schemes do not degrade search performance, and might improve it in some cases, their use can be recommended. Scheme 1 or Scheme 2 can indifferently be chosen, as the supplied results seem quite similar. Note, however, that the monitoring approach cannot act as a replacement for finding adequate settings of SA parameters (like the cost function). At best, it may help not to waste test effort where search convergence is too slow. But the intrinsic difficulty of the landscape for SA search remains unchanged.

6 Conclusion

As argued by (Harman and Jones, 2001), the development of empirical studies is a crucial step to establish a body of knowledge in search-based software engineering. This paper is intended to contribute to this step. We adopted a *measurement* perspective, building upon fundamental work on metaheuristic search and combinatorial problems. To the best of our knowledge, the related literature has been little explored by the testing community. We are not aware of other studies investigating measures to tune search-based test data generation.

Our empirical study involved a testing problem (issued from previous work on testing control systems), as well as three simple problems chosen to experiment with a variety of landscapes. This allowed us to consolidate our approach and investigate whether consistent results could be obtained for several examples. Work was focused on simulated annealing search, and was driven by the two constraints of our original testing problem: 1) we cannot afford a large number of system executions and 2) sub-optimal solutions are useless with respect to the test objective. We believe these constraints are shared by many other testing problems. One characteristic of search-based test data generation is that the cost of a candidate solution has to be evaluated by supplying it to the system under test. Except for unit testing of simple functions, this evaluation may become very computationally costly. Moreover, as soon as the search process is intended to serve a test coverage objective (e.g., activate a program branch, or trigger an exception), sub-optimal solutions are not of interest. Hence, it is hoped that the retained approach could be relevant to a number of other testing problems.

We use the diameter and autocorrelation measures to study the adequacy of alternative settings of the cost function and neighborhood operator. These measures can only have a limited explanatory power, but are still useful to reject inadequate landscapes and retain those that are reasonably well suited to SA search. In our study, the approach allowed the Boiler problem to be successfully revisited, yielding a better cost function than the ones obtained by *ad hoc* trials. Our future work will consider additional empirical studies to refine the criteria for accepting a landscape. In particular, the determination of typical threshold values for the current measures requires more investigation.

The diameter and autocorrelation measures characterize the *search landscape*, which does not include the search process. It would be more interesting to work on the *process landscape*, since ultimately we are only interested in the search process efficiency. However, characterization of the process landscape is much more difficult, and still remains an open issue for testing problems. Up to now, our work on the process landscape has addressed less ambitious issues, using two alternative approaches.

The first one, followed in (Abdellatif-Kaddour et al., 2003b), consisted in modifying the core SA algorithm to tailor it for the problem under consideration. At the expense of *ad hoc* trials, we obtained a specific SA variant that worked well on the Boiler example. However, this variant is not expected to be reusable. Indeed, additional experiments in (Abdellatif-Kaddour, 2003c) showed that it poorly performs on the QAP and calendar problems.

The second approach, explored in this paper, was intended to be less specific. It consists in monitoring search convergence, so as to stop runs that are unlikely to succeed within the allowed number of iterations, and restart the search from another point. The proposed monitoring schemes supplied consistent results for the four case studies we experimented with. The schemes are not costly to implement, and may improve search performance while having a neutral effect in the worst case. We will conduct further experiments to confirm these results. In particular, the relevance of our schemes relies on the assumption that the GRBS measure evolves as shown in Figure 3, with possibly different phase durations. We will study whether GRBS evolution could exhibit different patterns for other instances of testing problems (e.g., exhibit peaks, ...).

Finally, we hope that the first results presented in the paper will encourage more experimental studies based on the measurement of testing landscapes. As a first step, the proposed approach could provide a useful framework to revisit existing case studies.

Acknowledgements

The authors would like to thank the anonymous referees for their detailed and helpful comments on a previous version of this paper.

References

- Abdellatif-Kaddour, O., Thevenod-Fosse, P., and Waeselynck, H. 2003a. Property-Oriented Testing: A Strategy for Exploring Dangerous Scenarios. *Proc. ACM Symposium on Applied Computing (SAC'2003)*, Melbourne, USA, 1128-1134.
- Abdellatif-Kaddour, O., Thevenod-Fosse, P., and Waeselynck, H. 2003b. An Empirical Investigation of Simulated Annealing Applied to Property-Oriented Testing. *Proc. ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'03)*, Tunis, Tunisia.
- Abdellatif-Kaddour, O. 2003c. Property-Oriented Testing of Control Systems: Stepwise Construction of Test Scenarios Generated by Simulated Annealing Search. Doctoral Dissertation, Polytechnic National Institut of Toulouse, France, LAAS Report n° 03573. (In French).

Abrial, J.-R., Börger, E., and Langmaar, H. (Eds) 1996. *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*. Springer Verlag.

Adamopoulos, K., Harman, M., and Hierons, R.M. 2004. How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution. *Proc. Genetic and Evolutionary Computation Conference (GECCO 2004)*, LNCS 3103, Springer Verlag, 1338-1349.

Angel, E. and Zissimopoulos, V. 1998. Autocorrelation Coefficient for the Graph Bipartitioning Problem. *Theoretical Computer Science*, 191: 229-243.

Angel, E. and Zissimopoulos, V. 2000. On the Classification of NP-Complete Problems in terms of their Correlation Coefficient. *Discrete Applied Mathematics*, 99(1-3): 261-277.

Belaidouni, M. and Hao, J.K. 2000. Landscapes of the Maximal Constraint Satisfaction Problem. *Proc. 4th European Conference on Artificial Evolution (EA'99)*, LNCS 1829, Springer Verlag, 244-255.

Belaidouni, M. 2001. Metaheuristics and Search Landscapes. Doctoral Dissertation, University of Angers, France. (In French).

Belaidouni, M. and Hao, J.K. 2002. SAT, Local Search Dynamics and Density of States. *Proc. 5th European Conference on Artificial Evolution*, LNCS 2310, Springer Verlag, 192-204.

Clarke, J., Dolado, J.J., Harman, M., Hierons, R., Jones, B., Lumkin, M., Mitchell, B., Mancoridis, S., Rees, K., Roper, M., and Shepperd, M. 2003. Reformulating Software Engineering as a Search Problem. *IEE Proceedings Software*. 150(3): 161-175.

Connolly, D.T. 1990. An Improved Annealing Scheme for the QAP. *European Journal of Operational research*. 46(1): 93-100.

Eremeev, A.V. and Reeves, C.R. 2003. On Confidence Intervals for the Number of Local Optima. *Proc. EvoWorkshops 2003*, LNCS 2611, Springer Verlag, 224-235.

Frank, J., Cheeseman, P. and Stutz, J. 1997. When Gravity Fails: Local Search Topology. *Journal of Artificial Intelligence Research*. 7: 249-281.

Gross, H.G., Jones, B. and Eyres, D.E. 2000. Structural Performance Measure of Evolutionary Testing Applied to Worst-Case Timing of Real-Time Systems. *IEE Proceedings Software*. 147(2): 161-175.

Harman, M. and Jones, B.F. 2001. Search-Based Software Engineering. *Information and Software Technology*, 43(14): 833-839.

Holland, J. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hordijk, W. 1996. A measure of landscapes. *Evolutionary Computation*. 4(4): 335-360.

- Jones, T. and Forrest, S. 1995. Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. *Proc. Int. Conf. on Genetic Algorithms (ICGA'03)*, 184-192.
- Jones, B.F., Sthamer, H-H. and Eyres, D.E. 1996. Automatic Structural Testing Using Genetic Algorithms. *Software Engineering Journal*. 11(5): 299-306.
- Kirkpatrick, S., Gellat, C.D., and Vecchi, M.P. 1983. Optimization by Simulated Annealing. *Science*. 220(4598):671-680.
- Lammermann F. and Wegener J. 2005. Test-Goal-Specific Termination Criteria for Evolutionary White-Box Testing by Means of Software Measures, *Proc. 6th Metaheuristics International Conference (MIC'2005)*.
- Lundy, M. and Mees, A.I. 1986. Convergence of an annealing algorithm. *Mathematical Programming*. 34(1): 111-124.
- Maniezzo, V., Dorigo, M. and Colomi, A. 1995. Algodesk: an Experimental Comparison of Eight Evolutionary Heuristics Applied to the Quadratic Assignment Problem. *European Journal of Operational research*. 81(1): 188-204.
- McMinn, P. 2004. Search-Based Software Test Data Generation: A Survey. *Software Testing, Verification & Reliability*. 14(2): 105-156.
- Merz, P. and Freisleben, B. 2000. Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem. *IEEE Trans. on Evolutionary Computation*. 4(4): 337-352.
- Michael, C.C., McGraw, G., Schatz, M.A. 2001. Generating Software Test Data by Evolution. *IEEE Trans. on Software Engineering*. 27(12): 1085-1110.
- Nugent, C.E., Vollman, T.E. and Ruml, J. 1968. An Experimental Comparison of Techniques for the Assignment of Facilities to Locations. *Operations Research*, 16: 150-173.
- O'Sullivan, M., Vössner, S. and Wegener, J. 1998. Testing Temporal Correctness of Real-Time Systems – a New Approach using Genetic Algorithms and Cluster Analysis. *Proc. 6th European Conference on Software Testing, Analysis & Review (EuroSTAR 1998)*.
- Pargas, R.P., Harrold, M-J. and Peck, R.R. 1999. Test Data Generation Using Genetic Algorithms. *Software Testing, Verification & Reliability*. 9(4): 263-282.
- Rayward-Smith, V.J., Osman, I.H., Reeves, C.R. and Smith, G.D. 1996. *Modern Heuristic Search Methods*. Wiley.
- Rosé, H., Ebeling, W. and Asselmeyer, T. 1996. The density of States – a Measure of the Difficulty of Optimisation Problems. *Proc. Parallel Problem Solving from Nature (PPSN IV)*, LNCS 1141, Springer Verlag, 208-217.

- Schultz, A.C., Grefenstette, J.J. and De Jong, K.A. 1995. Learning to Break Things: Adaptive Testing of Intelligent Controllers. *Handbook on Evolutionary Computation*, chapter G3.5. IOP Publishing Ltd. and Oxford University Press.
- Stadler, P.F. and Schnabl, W. 1992. The Landscape of the Traveling Salesman Problem. *Physics Letters A*, 161(4): 337-344.
- Taillard, E.D. 1991. Robust Tabu Search for the Quadratic Assignment Problem. *Parallel Computing*. 17(4&5): 443-455.
- Tracey, N., Clark, J., Mander, K. and McDermid, J. 1998a. An Automated Framework for Structural Test-Data Generation. *Proc. 13th IEEE Conference on Automated Software Engineering (ASE)*. Hawaii, USA, 285-288.
- Tracey, N., Clark, J., and Mander, K. 1998b. Automated Program Flaw Finding Using Simulated Annealing. *Proc. ACM Int. Symp. on Software Testing and Analysis (ISSTA'98)*. Clearwater Beach, Florida, USA, 73-81.
- Tracey, N., Clark, J., Mander, K. and McDermid, J. 2000a. Automated Test-Data Generation for Exception Conditions. *Software – Practice and Experience*. 30(1): 61-79.
- Tracey, N. 2000b. A Search-Based Automated Test Data Generation Framework for Safety-Critical Software. PhD Dissertation, University of York, UK.
- Vanneschi, L., Tomassini, M., Collard, P. and Clergue, M. 2003. Fitness Distance Correlation in Structural Mutation Genetic Programming. *Proc. Europ. Conf. on Genetic Programming (EuroGP'03)*, LNCS 2610, Springer Verlag, 455-464.
- Wegener, J., Sthamer, H.H., Jones, B.F. and Eyres, D.E. 1997. Testing Real-Time Systems Using Genetic Algorithms. *Software Quality Journal*. 6(2): 127-135.
- Wegener, J., Buhr, K. and Pohlheim, H. 2002. Automatic Test Data Generation for Structural Testing of Embedded Software Systems by Evolutionary Testing. *Proc. Genetic and Evolutionary Computation Conference (GECCO-2002)*. New York, USA, 1233-1240.
- Wegener, J. and Buehler, O. 2004. Evaluation of Different Fitness Functions for the Evolutionary Testing of an Autonomous Parking System. *Proc. Genetic and Evolutionary Computation Conference (GECCO-2004)*. Springer Verlag, LNCS 3103, 1400-1412.
- Weinberger, E. 1990. Correlated and Uncorrelated Landscapes and How to Tell the Difference. *Biological Cybernetics*. 63: 325-336.
- Yokoo, M. 1997. Why Adding More Constraints Makes a Problem Easier for Hill-Climbing Algorithms: Analyzing Landscapes of CSPs. *Proc. Int. Conf. on Principles and Practice of Constraint Programming (CP'97)*. LNCS 1330, Springer Verlag, 356-370.

Appendix A – The Boiler Case Study

The steam boiler problem was proposed as a challenge for formal methods for safety-critical control systems (Abrial et al. 1996). The target system consists of a plant, a control program and a message transmission system connecting them. The plant has the following physical units: the steam-boiler, 4 pumps to provide water, 4 pump controllers indicating whether or not water is flooding from the pump, a sensor to measure the level of water in the boiler, a sensor to measure the quantity of steam which comes out, an operator desk. The program serves to control the level of water. Its functional requirement is to maintain the level between two nominal values N_1, N_2 . Its safety requirement is to maintain the level between two safety limits M_1, M_2 , where $M_1 < N_1$ and $M_2 > N_2$, otherwise the boiler explodes. The program has a cyclic behavior, and executes the following actions every five seconds:

- Reception of messages from the plant. There are 33 possible messages, including sensor data (e.g., LEVEL(v), where v measures the water level in the boiler) or messages from the operator desk (e.g., STOP, when an emergency shutdown is requested by the operator).
- Analysis of the information received from the plant. The control program has to update its view of the physical environment. As any device may become faulty at any time, the program cannot trust the received data. Plausibility and consistency checks are performed. The checks are based on the view elaborated at the previous cycle, as well as on expectations on the system dynamics. Depending on the check results, the program decides which devices are working correctly, and computes a new estimation range for the water level.
- Synthesis of messages and transmission to the plant. The program determines its operating mode, and sends it to the plant. The *emergency stop* mode is sent if the program considers that safety is endangered. The other modes are *initialization*, *normal*, *degraded* or *rescue* depending on the (perceived) states of the devices. Failure detection messages may also be sent to inform the operator of the faulty devices. Commands to open or close the pumps are sent to control the level. Overall, 27 messages are possible.

A more detailed description of the case study can be found in (Abrial et al. 1996). Many contributions were made, some of them proposing an implementation of the control program. A plant simulator was provided by the FZI to allow contributors to run and test their implementation.

Our test experiments involved the implementation proposed by J-R. Abrial and Steria. It is a C program of approximately 4800 lines of code (including comments) that can be interfaced with the FZI simulator. The test environment is shown in Figure A.1.

The tested system is composed of the control program and the plant simulator. Once started, the system autonomously operates. The closed-loop control is then perturbed by simulating the occurrence of faults in the devices, at chosen cycles of operation: our test inputs are timed fault commands for the simulator. For a time window of n cycles, and a maximum of 10 faults (affecting the 4 pumps, the 4 pump controllers, the water sensor and the steam sensor), the number of possible input sequences is:

$$\sum_{f=1}^{10} \binom{10}{f} n^f.$$

When an input sequence is supplied to the system, an execution trace is recorded. It contains the data needed to identify a boiler explosion or a dangerous situation. An explosion is easily identified because the simulator notifies it. Dangerous situations require more analysis. Whether the control program correctly diagnosed the faulty devices is determined from the failure detection messages sent to the plant. Whether the program correctly estimated the water level is determined by comparing its estimation range with the “actual” value in the simulator.

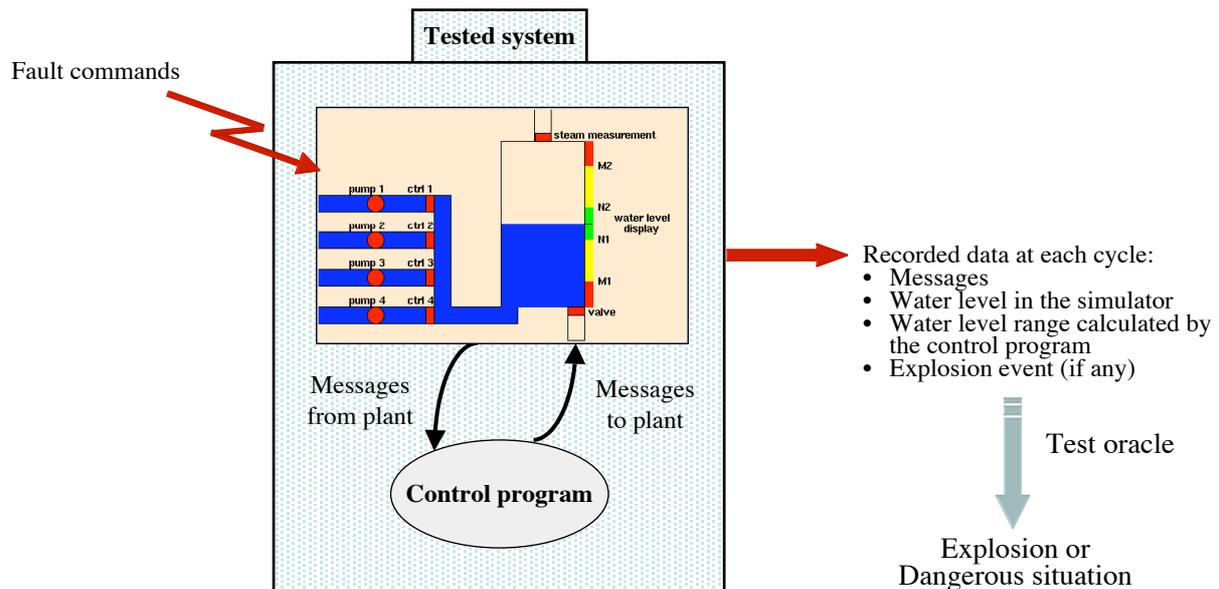


Figure A.1 – Test environment used in our experiments

Appendix B – Landscapes for the Boiler Problem

There are 2 landscapes, by combining 2 cost functions x 1 neighborhood operator.

B.1 Cost1

Test sequences fulfilling the test objective (i.e. yielding a boiling explosion or a dangerous situation) are assigned a zero cost. Other test sequences are assigned a positive cost, equals to the minimum of penalties associated with the three dangerous situations mentioned in Section 2.1. Intuitively, cost f_i is intended to penalize test sequences that are not “stressful” with respect to dangerous situation i .

- Cost f_1 is related to the control program not detecting the failure of a device. When none of the devices is faulty, this situation cannot occur: hence f_1 is assigned a maximum penalty K_1 . Then, we let f_1 decrease as the number of faulty devices increases.
- Cost f_2 is related to the control program wrongly detecting the failure of a device that is non-faulty. The Boiler environment includes 10 devices (4 pumps, 4 pump controllers, 1 water level sensor, 1 steam sensor). When the test sequence makes all these devices faulty, wrong detection cannot occur: f_2 is assigned a maximum penalty K_2 . Then, we let f_2 decrease as the number of non-faulty devices ($10 - nbFaultyDevices$) increases.
- Cost f_3 is related to bad estimation of the water level. At each cycle, the control program computes a lower bound qc_1 and upper bound qc_2 of the water level ($qc_1 < qc_2$). A dangerous situation is reached when the actual water level (observed from the boiler simulator) is outside the range $[qc_1, qc_2]$. Sequences not fulfilling this objective are considered all the more stressful as the uncertainty about the actual level is high, i.e. f_3 is decreasing with $(qc_2 - qc_1)$. In this way, the maximum penalty is when the estimation range is both correct (no dangerous situation) and accurate at the highest possible precision.

```

IF (boiler explosion OR dangerous situation observed) THEN
    Return (0)
ELSE
    Return (Min (f1, f2, f3))
Where:

$$f_1 = \frac{K_1}{nbFaultyDevices + 1}$$


$$f_2 = \frac{K_2}{10 - nbFaultyDevices + 1}$$


$$f_3 = \frac{K_3}{qc_2 - qc_1}$$

ENDIF

```

The experiments reported in Section 2.3 involved the following values for constant parameters K_i : $K_1 = K_2 = 250$, $K_3 = 4000$. Calibration was performed empirically, with the aim of providing similar variation ranges for all f_i . This yielded $[22, 250]$ for f_1 and f_2 , $[7, 160]$ for f_3 .

B.2 Cost2

The positive costs returned by Cost2 are primarily based on the consideration of water level estimation. The first term corresponds to function f_3 defined for Cost1. It rewards test sequences that increase the control program's uncertainty about the water level. The second term is intended to discriminate between test sequences yielding the same uncertainty. It favors scenarios that make the actual level q get close to the safety limits M_1 and M_2 (the boiler explodes when the water level gets lower than M_1 or greater than M_2).

```

IF (boiler explosion OR dangerous situation observed) THEN
    Return (0)
ELSE
    Return  $\left( \frac{K_4}{qc_2 - qc_1} + K_5 \cdot \text{Min}(|q - M_1|, |q - M_2|) \right)$ 
ENDIF

```

The experiments reported in Section 2.3 involved the following values for constant parameters K_i : $K_4 = 10,000$ and $K_5 = 0.2$. These values were intended to give higher weight to the first term than to the second one. The variation ranges are respectively $[20, 400]$ and $[0, 70]$.

B.3 Neighborhood Operator

The neighbors of a test sequence T are obtained from T by either: (i) changing the date of one fault of T; or (ii) adding one fault to T at an allowed date; or (iii) removing one fault from T. The allowed dates depend on the space explored by the test strategy. The experiments reported in Section 2.3 correspond to the space of sequences with prefix Steam_fault(3) and additional faults occurring at cycles $[4, 8]$. The neighborhood operator is applied only to these additional faults. For example, starting from Steam_fault(3), Pump2_fault(5), WaterLevel_fault(8) it may generate:

- Steam_fault(3), Pump2_fault(4), WaterLevel_fault(8) by changing the date of Pump2-Fault.
- Steam_fault(3), Pump1_fault(5), Pump2_fault(5), WaterLevel_fault(8) by adding a Pump1-Fault.
- Steam_fault(3), Pump2_fault(5) by removing the water level fault.

Overall, there are $8 + 35 + 2 = 45$ neighbors of the original sequence.

Appendix C – Landscapes for the Cal 1 problem

There are 24 landscapes, corresponding to the combinations of 4 cost functions x 6 neighborhood operators.

C.1 Cost Functions

	Definition
Cost1	If $(day, month, year) = (31, 12, 2000)$ then $Cost1 = 0$ Else $Cost1$ is randomly chosen in $[1, 1000]$
Cost2	$Cost2 = 31-day + 12-month + 2000-year + K_{day} + K_{month} + K_{year}$
Cost3	$Cost3 = 31-day + 31* 12-month + 365* 2000-year $
Cost4	Exact number of days between date $(day, month, year)$ and date $(31, 12, 2000)$

Cost1 assigns a random cost value to any solution, except triplet $(31, 12, 2000)$ which is assigned a zero cost.

Cost2 is inspired from the cost function used by (Tracey, 200b). It considers each triplet parameter, and measures its deviation from the target value in the optimal triplet. For example, $|31-day|$ measures deviation of the *day* parameter value, while $|2000-year|$ measures deviation of the *year* parameter value. Furthermore, as soon as deviation of parameter *i* is not zero, a constant penalty K_i is added, where *year* deviation is more penalized than *day* or *month* one:

- If $day = 31$ then $K_{day} = 0$ else $K_{day} = 10$
- If $month = 12$ then $K_{month} = 0$ else $K_{month} = 10$
- If $year = 2000$ then $K_{year} = 0$ else $K_{year} = 100$

The total cost is then the sum of individual costs for *day*, *month* and *year*. The variation range of *Cost2* is $[0, 1167]$.

Cost3 and *Cost4* make the cost value depend on the number of days between the current date and the optimal one. *Cost3* corresponds to a crude approximation of this number of days, while *Cost4* is exact. The variation ranges are respectively $[0, 365371]$ and $[0, 364999]$.

Note that the cost functions designed for Cal 2 are similar to the ones for Cal 1. They have just been adapted to account for the fact that Cal 2 has two optimal solutions. For example, *Cost4* computes the minimal number of days between the current date and any one of the two optimal dates.

C.2 Neighborhood Operators

	Definition	$ N $ = maximal number of neighbors a date may be connected to
N1	Any date obtained by letting one, two or three triplet parameters (<i>day, month, year</i>) vary ± 1 , modulo the maximal parameter value.	26
N2	Same as N1, without modulo (e.g. a December date is no more connected to a January one).	26
N3	Any date no more than 15 days apart from the current date, modulo the solution space boundaries, that is, (1, 1, 1900) and (31, 12, 3000) are connected.	30
N4	Same as N3, without modulo.	30
N5	Any date no more than 400 days apart from the current date, modulo the solution space boundaries.	800
N6	Same as N5, without modulo.	800

Intuitively, *N1* and *N2* should be well-suited to cost function *Cost2*, while *N3-N6* are more in the spirit of *Cost3* and *Cost4*.

Note that the neighborhood operators designed for Cal 2 are exactly the same as the ones for Cal 1.