



ReSIST: Resilience for Survivability in IST

A European Network of Excellence

Contract Number: 026764

Deliverable D11: Support for Resilience-Explicit Computing - first edition

Report Preparation Date: September 2007

Classification: Public

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
Fundação da Faculdade de Ciências da 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

Deliverable D11: Support for Resilience-Explicit Computing - first edition

Co-ordinator: Tom Anderson⁴

Editors: Zoe Andrews⁴, John Fitzgerald⁴

Contributors in ReSIST: Tom Anderson⁴, Zoe Andrews⁴, Cinzia Bernardeschi⁵, John Fitzgerald⁴, Michael Harrison⁴, Marc-Olivier Killijian³, Imre Kocsis¹, Melinda Magyar¹, Istvan Majzik¹, Zoltan Micskei¹, Ian Millard⁷, Nick Moffat⁶, Peter Popov², Peter Ryan⁴, Robert Stroud⁴

External contributor: Giovanna Di Marzo Serugendo (Affiliate Researcher, Birkbeck College)

Comments: ReSIST Res-Ex SIG, ReSIST Executive Board

¹Budapest University, ²City University, ³LAAS-CNRS, ⁴Newcastle University,

⁵University of Pisa, ⁶QinetiQ, ⁷Southampton University

Contents

1	Introduction	6
1.1	Resilience-Explicit Computing	6
1.2	Approach	8
1.3	Report Structure	9
2	First Edition Resilience Mechanisms	9
2.1	Cooperative Backup	9
2.2	Consensus Mechanisms	10
2.3	ModelWorks	11
2.4	Robust Re-Encryption Mixes	11
2.5	Dynamic Function Allocation	12
2.6	Supervisory Systems	13
2.7	Autonomic Computing Architecture	13
2.8	Robustness Testing	13
2.9	Model-based Stochastic Dependability Evaluation Tool	14
2.10	N-Version Programming/1/1	14
2.11	Recovery Blocks/1/1	15
2.12	N-Self-Checking Programming/1/1	15
2.13	Classification of First Edition Mechanisms	15
3	Interfaces for Adding/Viewing Res-Ex Mechanism Descriptions	19
3.1	Accessing Mechanism Descriptions	19
3.1.1	Human-Readable Mechanism Descriptions	19
3.1.2	Triple Browser	20
3.1.3	SPARQL Interface	22
3.2	Adding Mechanism Descriptions	23
3.2.1	Creating, Saving and Editing Mechanism Descriptions	23
3.2.2	Entry Types	23
3.2.3	Common Problems	28
3.3	Mechanism Description Fields	30
3.3.1	Overview	30
3.3.2	Classification	31
3.3.3	Further Details	31
3.3.4	Prerequisites	32
3.3.5	Resilience Metadata	32
3.3.6	Supporting Documents, if applicable	33
3.3.7	Research Areas	34
4	RKB: Overview and Res-Ex Extensions	34
4.1	RKB Technologies	34
4.2	RKB Content	35
4.3	Res-Ex Ontology	36
5	Related Work	38
5.1	Multi-Agent Systems	38
5.2	Web Services	39
5.3	GRID computing	39
5.4	Dynamic reconfiguration	40
5.5	Component-Based Software: selecting components	41
6	Evaluation and Future Work	41
6.1	Exploitation of Metadata and Mechanisms	42
6.2	Second Edition Mechanisms	44

6.2.1	Need for a Second Edition	44
6.2.2	Potential Second Edition Mechanisms	45
6.3	Entry Interface	46
6.4	RKB Explorer Interface and Res-Ex Ontology	47
6.5	Concluding Remarks	48
References		49
Appendix A: Res-Ex Case Study in Overview		53
A.1	Introduction	53
A.2	Decision Making with Metadata	53
Appendix B: Completeness of First Edition Mechanism Descriptions		56
Appendix C: Competency Questions		58

1 Introduction

This report forms part of Deliverable D11, from ReSIST Work Package 1 (Integration Technologies). In accordance with the Programme of Work, the deliverable is:

support for resilience-explicit computing (first edition), prepared by task IT-T2: This deliverable will demonstrate how resilience mechanisms can be represented in terms of resilience metadata and will describe the extended resilience ontology, with reference to the content and organisation of the validated knowledge base.

The deliverable has two components. First, the extended ReSIST Resilience Knowledge Base (RKB) contains “first edition” descriptions of resilience mechanisms in terms of resilience metadata, based on an extended resilience ontology. Second, this report provides an overview of the mechanism descriptions, the interfaces and mechanisms available for contributing further content, and the potential for extending the repertoire of mechanism and metadata descriptions in future.

1.1 Resilience-Explicit Computing

A long-term goal of current research is the provision of methods and tools that support the development and operation of ICT systems that exhibit predictable levels of resilience. Current system development methods rarely treat resilience-related information explicitly, making it difficult to predict system resilience and identify weaknesses. By contrast, in a *resilience-explicit* (Res-Ex) approach, information about the resilience-related properties of components and infrastructure are stated explicitly in the form of *metadata* published by components themselves, or by observers. Such metadata can be used at design-time to inform the choice of design patterns and development tools, or at run-time to tune or reconfigure, maintaining resilience. We use the term *resilience-explicit computing* to encompass both the design-time and run-time use of resilience-related metadata.

We use the term *metadata* to refer to information on which human or machine decision-makers act in order to maintain or enhance a system’s resilience. We use the “meta-” prefix in order to differentiate this from the data over which a system is performing its functionality. Examples of metadata include: a person’s workload in a socio-technical system, descriptions of known failure modes declared in the functional specification of a component, or historical availability statistics. Metadata could also be declared at different levels, such as components, the whole system or even for the user-interface of the system. We may even conceive of a market in trustworthy metadata, whereby metadata on service resilience might be provided by third parties and used to govern run-time selection of components and services.

In order to support machine-assisted decision-making, especially at run-time, it is necessary to develop languages for representing resilience metadata. Examples of representations include simple enumerations (e.g., component integrity levels), numeric representations (e.g., probabilities), or possibly formal logical descriptions (e.g., functional preconditions). Semantics are required for metadata so that analyses can be conducted consistently and with machine support. In particular, common semantics are required to ensure compatibility of metadata from heterogeneous sources (e.g., to ensure that metadata labelled “failure rate” from two different component providers are either interchangeable or convertible). The analyses that we

envisage going on at run-time or design-time as part of the decision to adapt or reconfigure may involve calculation over numeric metadata, logical deduction or, most likely, a mixture of the two.

As well as precise descriptions of metadata, Res-Ex computing requires that we have descriptions of the mechanisms that may be deployed or configured in order to meet a resilience target. We therefore use the term *resilience mechanism* to refer to a design pattern, technique or tool intended to improve system resilience. Examples include fault-tolerant architectural patterns (e.g., n-version programming) and development tools (e.g., robustness testing tools). In order to exploit resilience metadata in machine-supported decision-making, we require theories that describe the characteristics of the resilience mechanisms that may be deployed or configured within a system in terms of the relevant metadata.

We focus on the decisions to select a particular resilience mechanism from among alternatives and to instantiate or configure the mechanism for a specific application. Such decisions may be made statically, at design-time, or dynamically within a running system. In either case, in order to reach a resilience target, the decision-maker requires metadata about the characteristics (e.g., failure rates) of components, infrastructure and environment, and descriptions of the resilience mechanisms in terms of their effects on metadata, for example failure rate of a fault tolerant assembly in terms of failure rates of its components, or metadata generated by a robustness testing tool. The resilience mechanism descriptions may be combined with metadata to obtain a prediction of the consequences of a particular selection or configuration.

The goal of our work in ReSIST is to encourage the community to give descriptions of mechanisms and metadata that support this decision-making process. In particular, we wish to promote the contribution of mechanism descriptions in a form that enables automated analysis. There is currently very little support for gathering such descriptions or for making use of them. The descriptions of resilience mechanisms available to practitioners at present are deeply embedded in the scientific literature and are in many cases hard to extract. We wish to encourage researchers developing new mechanisms to give descriptions that help answer the question “What exactly does this mechanism achieve in terms of resilience?” We hope thereby to encourage research to evaluate existing and new mechanisms, and scholarship in codifying that information and making it available to practitioners. The work of ReSIST Task IT-T2 is to develop a means of recording descriptions of resilience mechanisms that are based on metadata and which integrate with the emerging Resilience Knowledge Base (RKB). This allows mechanism descriptions to be linked to other resilience knowledge through the emerging ontologies and through the research and training/education data embedded in the RKB.

A Scenario

In order to further clarify the resilience-explicit computing concept, consider a simple scenario (presented in more detail in Appendix A). A designer requires a system that tolerates one (sequential) hardware fault and/or one software fault. The designer has limited resources available and wishes to provide a cost effective solution. However, the system must also be as reliable as possible. The designer knows about three fault-tolerant architectures that would provide the necessary level of tolerance. These are, in our terms, *resilience mechanisms*:

- Recovery Blocks (RB/1/1)

- N-Version Programming (NVP/1/1)
- N-Self Checking Programming (NSCP/1/1)

Which of these mechanisms provides suitable cost and reliability levels? *Metadata* can be obtained for the three alternatives, including number of components, structural overheads, and operational time overheads in normal operation and when errors occur. For example, for RB/1/1, metadata includes¹:

Total number of variants required (= 2)

Total number of hardware components required (=2)

Ratio of Development and Maintenance Cost of fault Tolerant versus Cost of non-FT software (Min 1.33; Avg 2.17; Max 1.75)

Probabilities of detected and undetected failures on demand (as functions of probabilities of independent faults in components and decider).

These metadata are described in more detail in Appendix A, where a decision favouring RB on reliability and cost grounds is also illustrated. If other metadata, such as the run-time overheads when errors occur, are also taken into account NVP or NSCP may be preferable for different applications.

1.2 Approach

In order to make progress towards our goal of providing resilience-explicit guidance for the developer community, we aim to provide metadata-based descriptions of a large number and wide range of resilience mechanisms. We have begun this task by asking specialists across the ReSIST network to provide a preliminary and broad-ranging set of “first edition” descriptions. The mechanism descriptions provided are described in overview in Section 2.

The full first edition mechanism descriptions have been included in the on-line Resilience Knowledge Base (RKB) and are accessible to readers at <http://resist.ecs.soton.ac.uk/resex/>. The RKB is a key integrative technology contributed by ReSIST, gathering information on projects, publications, people, resilience mechanisms, educational materials and course descriptions. The addition of Res-Ex mechanism descriptions is part of the ongoing expansion of the value-added content of the RKB. Incorporating the mechanism descriptions requires utilisation of the existing ontological capability of the RKB, but expanding it to cover mechanisms and metadata via a Res-Ex ontology. More information on this aspect is included in Section 4.

Giving the RKB the capability of holding Res-Ex mechanism descriptions is not sufficient to support expansion of the collection. There must also be an interface whereby mechanism descriptions can be fed into the RKB and maintained once entered. A prototype of such an interface has been developed and used for recording the first edition mechanisms. Although this is a relatively mundane task, it is nevertheless a substantive issue because the interface must align with the ontology. Crucially, it must guide the creator of a mechanism description to answer the right questions in the right context so that they deliver the required information in the appropriate format. The interface and some of its rationale are considered in Section 3 (which is necessarily rather lengthy).

¹ These metadata are derived from the comparative study in [Laprie et al., 1990].

1.3 Report Structure

This report is a guide to the metadata-based first edition mechanism descriptions, the interface for viewing and editing them, and the RKB extensions to support such descriptions. The first edition mechanisms are each briefly described in Section 2. Full descriptions are in the on-line RKB (<http://resist.ecs.soton.ac.uk/resex/>); here we briefly comment on each mechanism's salient characteristics and issues that arose during the entry of its description via the Res-Ex interface to the RKB. A user guide to adding and viewing mechanism descriptions (Section 3) is followed by a brief discussion of the extensions to the underlying ontology of the RKB (Section 4). Looking forward, we relate the ReSIST Res-Ex work to research on run-time selection and configuration of components and mechanisms in Section 5. Finally, in Section 6 we discuss the potential exploitation of resilience metadata and mechanisms, evaluate the first edition Res-Ex RKB extensions and look forward to future work aimed at increasing the quality and breadth of metadata and mechanism descriptions.

2 First Edition Resilience Mechanisms

In this section, we briefly review the example mechanisms included in the first edition of the Res-Ex support embedded in the RKB. The mechanisms selected were initially offered by members of the Res-Ex SIG² and, later, by other ReSIST partners. We endeavoured to include as wide as possible a variety of mechanisms, including classical architectural mechanisms such as n-version programming, dynamic mechanisms such as dynamic function allocation and design-time tools such as ModelWorks. They represent contributions from each of the initial ReSIST Working Group areas in resilience building (Architectures, Algorithms, Socio-technical systems, Verification and Evaluation). Project partners were encouraged to use the new interface to develop and record “first edition” mechanism descriptions and to provide feedback on the process of doing so.

Descriptions of all the mechanisms listed below can be found in the RKB (<http://resist.ecs.soton.ac.uk/resex/>). Appendix B shows to what depth the first edition resilience mechanisms have been described by exhibiting the questions that have been answered for each mechanism.

It may be observed that there are more mechanism descriptions in the RKB than are listed here. This is because some of the first edition mechanism descriptions refer to related resilience mechanisms for which simple placeholders consisting of just a title and an overview have been created in the RKB.

2.1 Cooperative Backup

The primary objective of cooperative backup is to improve long-term availability of data produced by mobile devices. The idea is borrowed from peer-to-peer cooperative services: participating devices offer storage resources and doing so allows them to benefit from the resources provided by other devices in order to replicate their data. The described cooperative backup mechanism, which we call MoSAIC [Courtes et al., 2006], can leverage (i) excess storage resources available on mobile devices and (ii)

² The Res-Ex SIG is a special interest group on resilience-explicit computing consisting of ReSIST members and affiliate researchers.

short-range, high-bandwidth, and relatively energy-efficient wireless communications (Bluetooth, ZigBee, or Wi-Fi).

Participating devices discover other devices in their vicinity using a suitable service discovery mechanism and communicate through single-hop connections, thereby limiting interactions to small physical regions. Anyone is free to participate in the service and, therefore, participants have no prior trust relationship. When out of reach of Internet access and network infrastructure, devices meet and spontaneously form ad hoc networks which they can use to back-up data. Devices eventually send data stored on behalf of other devices to an agreed Internet-based store. Eventually, data owners may restore their data by querying the store.

Representing this mechanism in the resilience-explicit knowledge base was relatively easy as we have been studying its design and implementation for a long time. When necessary, one can consider this mechanism as mostly a composition of other mechanisms that need to be parameterized. For example, a potential decomposition of the cooperative backup mechanism into smaller components can be based on a resource discovery component, a trust and cooperation incentive mechanism, a proximity map. Additionally, we have conducted an extensive analytic evaluation of various parameters of the mechanisms, which has been very beneficial for expressing the mechanism's metadata [Courtès et al., 2007].

2.2 Consensus Mechanisms

The consensus problem [Pease et al., 1980] in distributed computing encapsulates the task of group agreement in the presence of faults. In particular, any process in the group may crash at any time. Consensus is fundamental to core techniques in fault tolerance, such as state machine replication. However, there are some difficulties in achieving consensus in the presence of faults under a particular set of system and failure assumptions:

- In a synchronous system, it is possible to solve the consensus problem using a Byzantine Agreement Protocol. However, in order to tolerate n Byzantine failures, it is necessary to have $3n+1$ processes.
- In an asynchronous system, it has been proved that is impossible to solve the consensus problem in general. However, a number of approaches have been proposed that either weaken the asynchrony assumption in some way, or else weaken the consensus property itself.

Expressing consensus as a resilience-explicit mechanism is non-trivial because it is not really a single mechanism, but rather the specification for a distributed problem that needs to be solved, plus a set of algorithms or protocols that solve the problem or a variant under a specific set of system and failure assumptions. The existing model of a resilient explicit computing mechanism is not rich enough to capture these various subtleties and relationships, but as future work it would be worth trying to tease out the distinction between a specification, a set of related implementations of the specification, and the system and failure assumptions that each implementation depends on.

The approach that has been adopted for the current version of the deliverable is as follows:

- A top-level description of a consensus mechanism has been provided, together with three more specific descriptions of particular consensus mechanisms ("BFT - Practical Byzantine Fault Tolerance" [Castro et al.,

1999], “Signal-On-Fail based consensus protocol” [Inayat et al., 2006] and “Sintra - Secure Intrusion-Tolerant Replication Architecture” [Cachin et al., 2000]).

- Each mechanism refers to the other mechanisms, and a special Consensus concept is introduced to link the specific instantiations of the consensus mechanism.
- The subtleties of the various system and fault models used by each mechanism are described under "Other Prerequisites" rather than as metadata. This is because it would be very difficult to capture them using the existing categories of metadata - clearly, more research is required into how to describe these assumptions more formally as metadata, but this task is left for the next deliverable.
- Many of the attributes of the various consensus mechanisms are the same, or could be inherited from the top-level description. However, since the current version of the deliverable does not support inheritance, these attributes have had to be entered manually, and a certain amount of iteration was necessary before all four descriptions were consistent.

2.3 ModelWorks

ModelWorks is a QinetiQ in-house formal modelling tool. It consists of a GUI front-end to (currently) two formal modelling components: the Dependability Library and support for Assumption-Commitment (AC) reasoning. The ModelWorks GUI includes an editor for building graphical system design models and specifying system properties. There is an automatic translation capability from system designs to formal CSP (Communicating Sequential Processes) models, which can then be analysed by external automated tools. A wide range of discrete distributed systems can be modelled, and analysed with respect to safety, availability and security properties.

The chief difficulty with the description activity was the question of how the mechanism should be viewed: does an analysis tool perform fault forecasting, fault detection or both? If one considers a process that includes “Analyse using the tool, then act on the results by fixing discovered (detected) faults”, then does this perform fault tolerance? One approach is to characterise the mechanism itself strictly, not any way in which one might use it in a containing ‘process mechanism’. Another is to characterise all ways in which it might be used. We could allow ourselves separate mechanisms a) “the tool” and b) “use the tool, then act on the results in some defined way” and describe these separately in terms of their direct application/benefit. Finally, precise and meaningful characterisation of tool effectiveness may only be possible with reference to standard benchmarks, which do exist currently.

2.4 Robust Re-Encryption Mixes

Re-encryption mixes are a mechanism for providing anonymity in voter-verifiable voting systems [Ryan et al., 2006]. In essence, voters are provided with unique “protected receipts” at the time of casting that carry their vote in encrypted form. All receipts should be posted to a secure Web Bulletin Board (WBB). Voters can confirm that their receipt is correctly posted. Re-encryption assumes that plain text is encrypted using a randomised public key algorithm. In effect, it re-randomises the encryption without changing the plaintext and the plaintext is not revealed during the process. A set of mix tellers perform re-encryption mixes: each teller takes in a batch

of receipts as posted to the WBB, transforms each by re-encryption and posts the resulting batch of re-encrypted terms to the next column of the WBB. This can be done as many times as required. Once a suitable number of such mixes have been performed (to achieve whatever level of defence in depth for ballot secrecy is deemed appropriate) and all the shuffles posted to the WBB, independent auditors perform a Partial Random Check on the posted information such that each transformation has a 50/50 chance of being audited. If the posted information passes the audits, decryption tellers take over the decryption of the now (multiply) shuffled ballots. Once the ballots have been decrypted, a universally verifiable count can be performed.

The main challenge with recording this mechanism was relating it to the dependability and security ontology, which currently lacks some concepts relevant to security or cryptography applications, for example authentication mechanisms, zero knowledge proofs and coercion resistance. The extension of the ontology with more detailed security-related concepts is being addressed in the Resilience Ontology (ResOn) Special Interest Group within ReSIST.

2.5 *Dynamic Function Allocation*

Two mechanisms are described under this heading. The first is a design process that involves deciding how to automate a control system in order to support the human operator within that system most effectively. The second is the result of the design process where a control system is designed to adapt to the current situation in order that the human operator can maintain control in the face of considerations such as workload or situation awareness that will affect the operator's resilient performance.

A control system consists of functions designed to achieve the various aspects of the control task. The system allocates its functions differentially at different levels of automation involving more or less participation by the human operator. A level of automation for a function may require the operator to carry out the function entirely, or to supervise the completion of the function with a power to interrupt, or to be unaware of the function entirely. The full range of automation options in relation to a human role is discussed in [Dearden et al., 2000]. Controlling the system will involve a combination of these executing functions, requiring different levels of operator control. It may involve different strategies, combining the use of functions in terms of procedures in different ways depending on different factors (for example, the check out operator in a supermarket may choose to help the customer to pack purchased items if too many items have piled up in the purchased hopper). The mechanism by which different automation choices are made may be controlled by the operator but it may be automatic and may involve a decision procedure that samples measures associated with a number of factors: time on task; error rate; physiological workload may be used for example and may be used in combination.

There are a number of reliability metadata, some of which are difficult to measure: error rates; human workload; situation awareness. These are distinct concepts from those found elsewhere in the resilience literature. Both the design process and adaptive automation mechanism are decision procedures involving a utility trade-off.

Recording the mechanism description raised several interesting challenges. First, it was important to clarify the distinction between the process of deciding how to perform a dynamic function allocation and the mechanism of dynamic function allocation itself. This suggests that contributors will benefit from improved guidance on what constitutes a mechanism. The examples provided to help enter the

information into the RKB were resilience “mechanisms” in a traditional sense, that is mechanisms invoked at run time in a target system (for example recovery blocks). This did not help to understand the information that was required for a resilience mechanism that was to be incorporated as part of a design process, the performance of dynamic function allocation. Second, regarding RKB data entry, it would have been beneficial to have the mechanism description at an earlier stage. A trigger on how to describe possible metadata would also have been helpful. Third, the required metadata did not already exist and so had to be entered into the taxonomy; it would also have been helpful to be able to record a concept hierarchy in the taxonomy.

2.6 Supervisory Systems

Supervisory systems are systems and architectures that, using agent based technology, periodically sample the state and non-functional properties of resources and services in a general purpose IT environment and forward this information to a central management service. The management service deals with the persistent storage, classification, correlation and visualization of measurements and events. Note that most supervisory systems provide only a toolset out of the box; customarily, configuration design is a fully-fledged project on its own. Technological approaches on the agent level, the implicit/explicit nature of the data metamodel and the extent to which a certain tool can be integrated into a full control loop account for the main factors distinguishing available frameworks from the point of view of resilience mechanisms.

In general, the description approach suited the mechanism quite well. However, identifying the threats addressed was not an easy task. Currently, IT supervisory systems generally do not use the well-established ontology of classic dependability; their common set of metaphors does not even distinguish faults, errors and failures. The distinction of monitoring for specific faults, errors or failures comes with the design of the supervisory configuration and is, consequently, application-specific.

2.7 Autonomic Computing Architecture

The Autonomic Computing Architecture mechanism is an architectural mechanism proposing a service-oriented architecture encompassing the notions of autonomic components (services) managing their own behaviour on the basis of pre-established policies [White et al., 2004]. The underlying service-oriented infrastructure supports service/policy discovery and binding among the different autonomic elements. It also provides specific elements that support autonomic components for reasoning, negotiation, and monitoring. This specific architecture follows the generic architectural blueprint for autonomic computing defined by IBM [IBM 2006].

This mechanism uses run-time monitoring to trigger dynamic reconfiguration on the basis of the policies. It then becomes difficult to identify how to describe the mechanism from a developer’s point of view, i.e., how to separate the metadata aspects used for identifying the mechanism at design time from those used by the mechanism during a run-time instantiation..

2.8 Robustness Testing

The goal of robustness testing is to generate and execute test cases to assess the robustness of a computer system, i.e., the degree to which the system operates correctly in the presence of exceptional inputs or stressful environmental conditions

[Micskei et al., 2006]. The approach of robustness testing is similar to functional "black box" testing, but it concentrates on the activation of potential robustness faults. To do this, exceptional inputs are generated on the basis of the system interface specification, and stressful environmental conditions are provided by (i) a workload that determines the utilization of the system and (ii) a fault-load that determines how faults are injected into the environment of the system (e.g., hardware, operating system, configuration options). The test outputs are evaluated looking for responses (including crash and timeout) that do not comply with the specification.

The metadata included in the questionnaire characterise robustness testing as a general process by providing the threats that are addressed, the knowledge and infrastructure requirements, the failure modes, and the type of this verification method. These metadata highlight prerequisites of robustness testing (e.g., the interface description to be used to generate exceptional values) and its role in increasing the resilience of a system. Note, however, that in the case of a functional testing approach like robustness testing, there are no clear quantitative measures (metadata) that can be used to compare this process with other potential testing processes.

If a concrete robustness testing tool (e.g., a test generator or test harness) were to be described, then the above metadata could be extended with metadata characterizing the concrete input and output formalisms, the resource requirements and the other peculiarities of the tool that implements the general process.

2.9 Model-based Stochastic Dependability Evaluation Tool

The model-based stochastic dependability evaluation tool [Majzik et al., 2007] constructs a mathematically precise dependability model (in the form of a stochastic Petri net) from a UML-based architecture model of the system, and evaluates the model to get system level dependability measures (like reliability and availability) using the local dependability parameters (like fault occurrence rate, error latency, repair delay) of system components.

Since this mechanism is implemented by a tool, the hardware and software requirements were defined easily. The underlying mechanism is a model transformation with two steps, so the description of this process needed more effort. The related concepts, metadata, ontology and publication had to be collected. The selection of the failure modes and the research interests was a more difficult task because there were several choices which are not independent. The same problem occurred in the context of threats addressed and research interests.

2.10 N-Version Programming/1/1

The N-Version Programming/1/1 (NVP/1/1) mechanism is a specific variant of a classical fault-tolerant architecture described in the n-version approach to fault-tolerant software [Avizienis 1985]. This variant, described by [Laprie et al., 1990], addresses hardware fault tolerance as well as software fault tolerance. It uses three diverse implementations of a software module, each of which runs on distinct hardware, and voting on the results to provide fault tolerance.

It was straightforward to describe this mechanism in the Res-Ex interface. However, the description would be improved if it were possible to use the ontology and interface to directly link separate items of metadata by mathematical formulae to create different composite metadata, or just provide a different view of the same

metadata. It was also quite challenging to decide on the failure modes of the mechanism and to elicit the required knowledge for using it.

2.11 Recovery Blocks/1/1

The Recovery Blocks/1/1 mechanism is a specific variant of the classical recovery block approach to error recovery and fault tolerance as described in [Horning et al., 1974]. The variant described here is that from [Laprie et al., 1990], which they call RB/1/1, and treats the recovery block as a mechanism expressed via recovery block syntax and implemented with support for backward recovery. The specific variant considered has two alternate blocks and also provides hardware fault tolerance by replicating the two blocks on a distinct hardware platform that runs in hot standby.

As with N-Version Programming/1/1, this mechanism was not overly difficult to describe in a resilience-explicit way. One issue that was raised when doing so was the importance of being clear about exactly what is being described. Different people have different interpretations about the scope of a mechanism; therefore, it is important to clearly state the scope within the mechanism description. The comments on describing N-Version Programming/1/1 also apply to this mechanism.

2.12 N-Self-Checking Programming/1/1

N-Self-Checking Programming provides fault tolerance through the use of two or more components, each with the ability to check their own dynamic behaviour, running in hot standby. Such self-checking may be carried out in a number of ways. In the specific variant considered here, N-Self-Checking Programming/1/1 [Laprie et al., 1990], there are two self-checking components. Each self-checking component has two diverse implementations of a software module and compares the results from these implementations to check its behaviour. Thus, there are in total four implementations of the software module, all of which are diverse.

This mechanism is closely related to N-Version Programming/1/1 and Recovery Blocks/1/1. Therefore, the reader is referred to the points raised previously about the ease of providing resilience-explicit descriptions of such mechanisms.

2.13 Classification of First Edition Mechanisms

The mechanism descriptions in this section are not presented in a structured way. Indeed, many classification schemes can be considered. Table 1 provides classification of the mechanisms according to a variety of key characteristics:

- The partner responsible for contributing the mechanism description.
- The mechanism objectives.
- Whether the mechanism is an architecture, a process or a tool.
- The development/operational phase during which the mechanism can be applied (design, development more generally or run-time).
- Whether the mechanism provides fault detection, fault forecasting, fault removal, and/or fault tolerance.
- The resilience-building technology (RBT) with which the mechanism is most closely associated (corresponding to ReSIST Working Groups):

- *Architecture* – resilience architecting and implementation paradigms.
- *Algorithms* – resilience algorithms and mechanisms.
- *Socio-Technical* – resilient socio-technical systems.
- *Verification* – methods and tools for verifying resilience.
- *Evaluation* – methods and tools for evaluating resilience.
- The resilience-scaling technologies with which the mechanism is most closely associated (corresponding to ReSIST Working Groups):
 - *Evolvability* – Resilience evolvability, maintaining resilience during activities such as upgrading, recovery and fault handling, adaptation and reconfiguration.
 - *Assessability* – Resilience assessability, the ability of a system to assess its correct functioning and quality of service delivered under both nominal and stressful conditions.
 - *Usability* – Resilience usability, achieving or assessing usability of systems, particularly ubiquitous ones. Helping users interacting with ubiquitous systems to understand the potential effects of their actions as well as preventing them from taking actions with unwanted and difficult to anticipate system-level effects.
 - *Diversity* – Resilience diversity, the use of components that can perform similar functions in the system context but differ in some essential aspect that affects their vulnerability.
- The types of system to/within which this mechanism can be applied.
- The main direct benefits of the mechanism, in terms of improved system resilience (in the case of run-time deployment mechanisms) or assurance of key system properties (in the case of pre-deployment analysis tools); the latter type of benefit could lead to improved resilience of whatever systems are deployed, through bug-finding and potential freeing up of resources for other development activities, by achieving assurance at reduced cost.

When selecting the first edition of mechanism descriptions, we aimed for coverage of the resilience building technologies identified in the ReSIST programme. Table 1 shows that we succeeded in achieving such coverage. This provided a collection of complementary mechanisms that highlighted the variety of approaches required for creating such a broad set of mechanism descriptions.

	Partner	Objectives	Category	Phase	Fault Actions	Res-Building Tech.	Res-Scaling Tech.	Systems	Direct Benefits
Consensus Mechanism	Newcastle	Group agreement in the presence of faults.	Architecture	Run-time	Forecasting	Algo	Divers	Distributed computer systems	Agreement among correct components
Dynamic Function Allocation	Newcastle	To support the human operator effectively and resiliently in carrying out their control tasks.	Architecture	Run-time	Tolerance	Socio	Assess	Human-machine control	Effectiveness (incl. efficiency) and resilience
N-Self-Checking Programming/1/1	Newcastle	To tolerate faults through the use of components with the ability to check their own dynamic behaviour.	Architecture	Run-time	Tolerance	Arch	Assess, Evolv, Divers	Systems with self-checking components	Fault tolerance
N-Version Programming/1/1	Newcastle	To utilise design diversity and voting in order to tolerate software faults	Architecture	Run-time	Tolerance	Arch	Assess, Evolv, Divers	Systems with diverse components	Software fault tolerance
Recovery Blocks/1/1	Newcastle	To provide backward recovery to isolated sequential programs	Architecture	Run-time	Tolerance	Arch	Assess, Evolv, Divers	Sequential programs	Error recovery
Robust re-encryption mixes	Newcastle	To provide ballot secrecy by anonymising ballot receipts.	Process, Architecture	Run-time	Tolerance	Algo	Assess	Ballots	Security, anonymity, auditability
Model based stochastic dep. evaluation tool	BUTE	Model-based evaluation of architectural alternatives from the point of view of availability and reliability.	Tool	Design	Forecasting (& Removal)	Eval	Assess	Distributed	Assurance of availability and reliability
Robustness testing	BUTE	To generate and execute test cases to assess the robustness of a computer system.	Tool/ Process	Development	Forecasting (& removal)	Verif	Divers/ Assess	ICT + stressful environments	Assurance of robustness
Supervisory Systems	BUTE	To support real-time monitoring and visualisation of state and non-functional properties of hardware, software and service components in general IT infrastructures.	Architecture	Run-time	Detection	Arch	Assess	General purpose IT	Real-time monitoring and visualisation
Cooperative Backup	LAAS	Long-term availability of data produced by mobile devices.	Architecture	Run-time	Tolerance	Arch/ Algo	Divers	Systems containing mobile devices	Long-term-availability, reduced local storage need
Modelworks	QinetiQ	To provide scalable dependability assessment of systems	Tool	Design	Forecasting (& Removal)	Eval	Assess	Distributed	Assurance of safety, liveness and security
Autonomic Computing Architecture	ReSIST affiliate researcher	To provide an architectural approach to autonomic computing: self-configuration, self-healing, self-protection, self-optimisation.	Architecture	Run-time	Tolerance, Removal	Arch	Evolv	Autonomic	Efficient and resilient sharing of resources

Table 1: Summary of First Edition Resilience Mechanisms

3 Interfaces for Adding/Viewing Res-Ex Mechanism Descriptions

We aim to encourage a wide range of contributors worldwide to provide mechanism descriptions. For this first edition, we therefore developed a web-based interface that contributors from across the project could use as a common way of entering mechanism descriptions into the RKB. The use of a common format facilitates comparison between mechanisms and ensures that key questions, for example, regarding resilience metadata, are answered. The fact that the interface connects mechanism descriptions directly into the RKB brings the benefit of lineage between mechanism descriptions and other RKB entities, including relevant people, publications and projects.

The interface has been implemented as an extension of a generic form-based interface developed for use with various ReSIST activities, which enables knowledge acquisition against an underlying ontology. Through the use of a configuration script that prescribes the type of input control to present and other details for each ontological concept, an interface is automatically generated which enables data input and subsequent editing by authorised users, combined with a simple read-only display of the data for public viewing. The resilience-explicit (Res-Ex) mechanisms interface can be found at <http://resist.ecs.soton.ac.uk/resex/>.

3.1 Accessing Mechanism Descriptions

There are three ways in which the mechanism descriptions held within the RKB can be accessed: a human-readable interface intended as the main means of browsing and updating descriptions; a tabular view of raw data (the Triple Browser) and a direct query mechanism (in SPARQL).

3.1.1 Human-Readable Mechanism Descriptions

The resilience-explicit computing mechanisms interface (shown in Figure 1) presents a list of known mechanisms, permitting public access to a human readable presentation of the information stored.

By clicking on the title of a mechanism, a simple page will be displayed detailing the properties and values constituting the mechanism description. Where possible, for externally referenced resources such as publications, direct links are made to online versions of those resources. Where these are not available, and for other resources such as the resilience metadata values, links are presented to a raw view of the underlying semantic information in the RKB Triple Browser (see Section 3.1.2).

For authenticated users, the list of mechanisms is augmented with additional options to permit editing or deletion of mechanism descriptions.

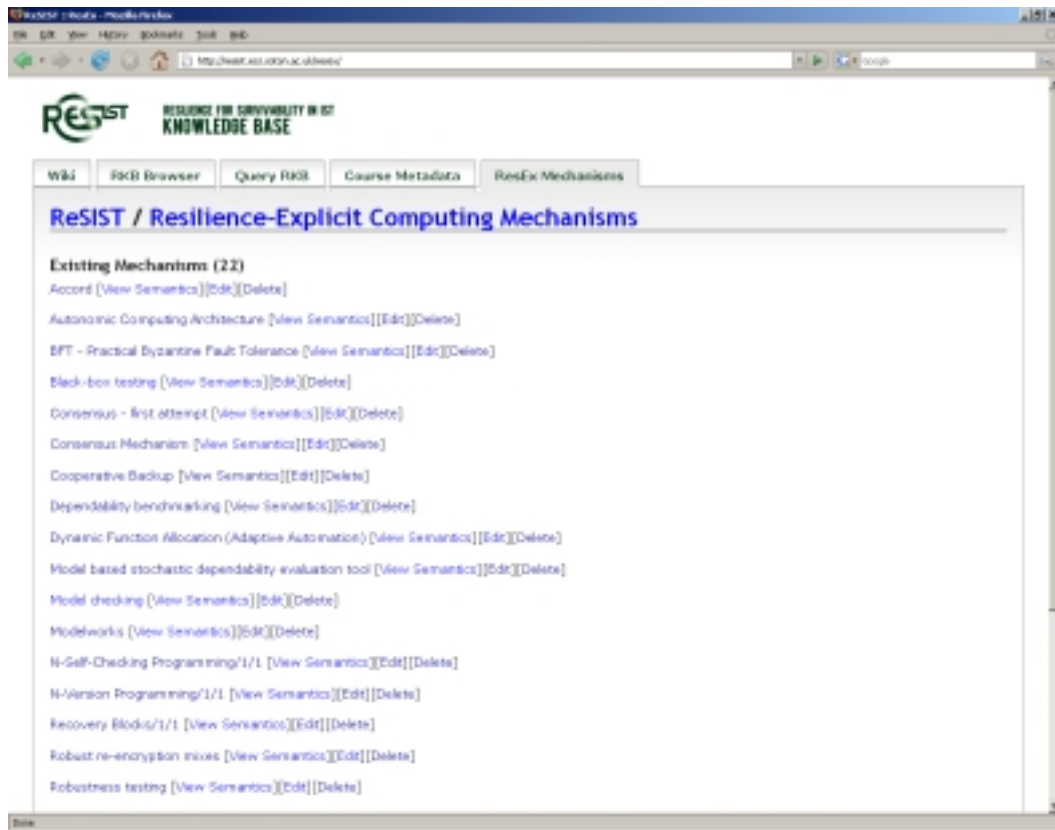


Figure 1: Res-Ex interface front page

3.1.2 Triple Browser

The RKB presents a generic web-based interface facilitating a tabulated view of the raw knowledge contained regarding a given resource. A simple search mechanism is implemented to enable users to find the URIs of semantic resources that have literal values which match a given string, in addition to allowing direct access to viewing the details of a specific URI. Again, this is a public access service, available at <http://resist.ecs.soton.ac.uk/browse/>.

When a particular URI is viewed in the triple browser, a table of data is presented in two halves, showing RDF triples (facts) from within the knowledge base. The upper half shows details for which the resource in question is the subject of a relation, i.e., facts in the format <URI> <predicate> <value>. Conversely, in the lower half, facts are shown in which the requested resource appears as the third term, i.e., <value> <predicate> <URI>. Figure 2 shows the triple browser view of information that can be found in the RKB about the SIG on resilience-explicit computing.

RKB Browser :: ReSIST Resilience-Explicit Computing

Identifiers in RKB "rdi"...

<http://resist.ecs.soton.ac.uk/wiki/nec>

Subject	Property	Object/Value	Source
ReSIST Resilience-Explicit Computing	akt:has-project-leader	John Fitzgerald	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Aad van Moorsel	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Algirdas Avizienis	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Andras Patencza	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Benedicto Rodriguez	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Birgit Pfitzmann	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Harigovind Ramasamy	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Holger Pfister	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Marc-Olivier Killian	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Michael Harrison	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Neil Henderson	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Nick Moffat	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Peter Popov	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Phil Cole	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akt:has-project-member	Zoe Andrews	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	akts:has-pretty-name	ReSIST Resilience-Explicit Computing	wiki-2004.rdf >>
ReSIST Resilience-Explicit Computing	rdf:type	akt:Project	wiki-2004.rdf >>

Alternative representations
 > RDF export
 > View in ReSIST Wiki (requires login)

Figure 2: Example triple browser page

The table of triple values is additionally augmented with a fourth column, identifying the source data from which each fact originated. Where resources have properties defining an `rdfs:label` value, these are used to display a "pretty" or human readable name instead of the raw URI. Similarly, predicates from known namespaces are abbreviated to a more readable format. Hovering the cursor over a URI, pretty name, or predicate will display the raw URI which is represented.

Users may navigate the entire knowledge base through the underlying connected graph representation of RDF. Each URI presented as a subject, relation or object within the table of triples is clickable, changing the focus of the triple browser to reflect that resource. For example, when viewing a person, the table will present facts such as `<person> <works-at> <organisation>`, and `<paper> <has-author> <person>`. Selecting the organisation would show not only further details of that resource, but also all other people in the knowledge base who work there in the form `<someone> <works-at> <organisation>`. Likewise, one can navigate from a person to a particular publication, then to details of a co-author, and see their publications.

While the Triple Browser is not the most elegant of interfaces, it does provide a very useful means of viewing the underlying data, and is reasonably intuitive for non-expert users.

3.1.3 SPARQL Interface

The RKB offers a direct query mechanism, through an implementation of the SPARQL Query Language for RDF³ at <http://resist.ecs.soton.ac.uk/sparql/>.

By submitting triple-pattern queries in the SQL-like SPARQL language, low level results may be obtained in both XML and tab-delimited ASCII formats. For example, the following query will return a set of results which consists of the identifier and name of all resources that are of type `resex:Resilience-Mechanism`.

```
SELECT * WHERE {
    ?id rdf:type resex:Resilience-Mechanism.
    ?id akt:has-title ?name
}
```

Figure 3 shows the results page that is returned on submitting this query to the SPARQL interface.

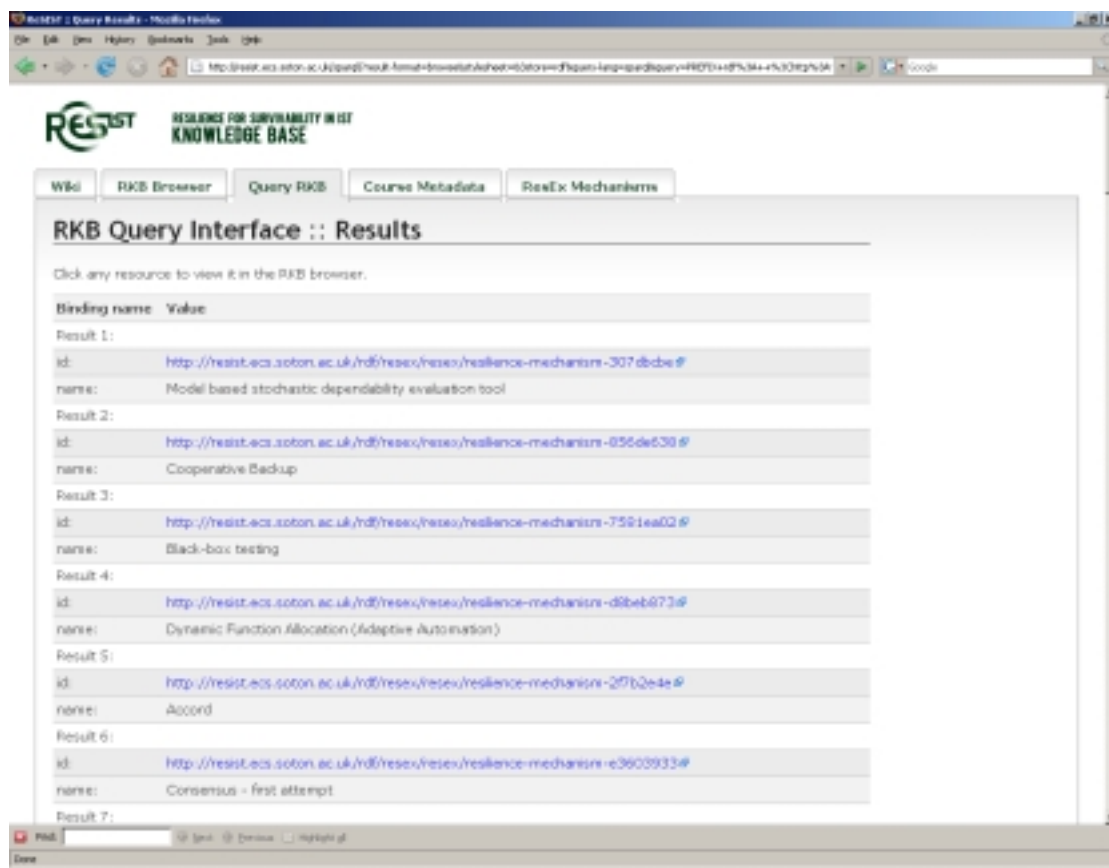


Figure 3: An example SPARQL interface results page

The provision of open access to the entire contents of the RKB through a standard interface enables external software processes to easily integrate with the knowledge available. This leads to the opportunity for systems to be developed which interrogate the RKB to discover resilience mechanisms with particular properties or those which

³ W3C Candidate Recommendation, 14 June 2007, <http://www.w3.org/TR/rdf-sparql-query/>

satisfy given constraints, as part of both development-time system proof and analysis tools, in addition to dynamic run-time configuration roles.

3.2 Adding Mechanism Descriptions

A forms-based interface (at <http://resist.ecs.soton.ac.uk/resex/>) has been developed for adding metadata-oriented descriptions of resilience mechanisms to the RKB. Users answer a series of questions about the resilience mechanism that is being described. This section provides an overview of how the user interacts with the interface to complete the forms. The details of the questions asked can be found in Section 3.3.

The interface has been developed for compatibility with version 2 of the Firefox browser⁴. It is therefore recommended that this browser is used when entering descriptions of resilience mechanisms.

It is important to be clear about exactly what resilience mechanism is to be described and at what level of specificity before beginning to enter metadata about it into the interface (see Section 3.2.3).

The page for editing mechanism descriptions includes an “e-mail us” link for reporting technical problems or suggesting improvements.

3.2.1 Creating, Saving and Editing Mechanism Descriptions

Users creating or editing a mechanism description must be known to the ReSIST wiki and log on in the normal manner. Once logged on, users navigate to the main interface page (<http://resist.ecs.soton.ac.uk/resex/>) to continue. The “click here” link under the “Add a new mechanism” heading opens a blank form for mechanism entry.

Mechanism descriptions can be saved and returned to at any time. Following the “Continue” links at the bottom of each page until the “Finish” button is clicked takes the user to a human readable overview of the information added. The mechanism will then appear in the list of mechanisms on the main page.

From the main page, an existing mechanism can be edited by clicking on the “Edit” link next to its title. This opens the form with the previously entered information already present and allows users to edit/delete information from, or add more information to, the mechanism description. These changes can be saved as described above.

3.2.2 Entry Types

Several different types of data entry are used. Although most are intuitive, we provide an overview of each of them below. The entry type of each field of the interface is given in Section 3.3.

Text Field

A text field allows the user to enter a line of free text (Figure 4 shows an example from the Res-Ex interface).

⁴ Available to download for free at <http://www.getfirefox.com/>

Name of the resilience mechanism (A title to identify your mechanism)	Recovery Blocks/1/1
--	---------------------

Figure 4: An example text field

Text Area

A text area allows multiple lines of free text to be entered (Figure 5).

Detailed Description (Either enter a detailed description of the mechanism here, should be detailed enough for the reader to be able to re-create the mechanism, or reference a paper with such text in below)	The information here applies to the specific variant of the mechanism RB/1/1, described in "Definition and Analysis of Hardware- and Software-Fault Tolerant Architectures". Thus we are treating the recovery block as a mechanism expressed via recovery block syntax and implemented with support for backward recovery. The specific variant considered, RB/1/1, has two alternate blocks.
---	--

Figure 5: An example text area

Multi Select

Items are selected by clicking on them; multiple items are selected by holding down Ctrl (or the apple key on an Apple Mac) whilst clicking on them. Ctrl-click toggles the selection status of the item clicked on. Further information about items in the list can be obtained by hovering the mouse pointer over them. Figure 6 shows an example of a multi select field in the Res-Ex interface.

Submitted by (The person(s) identified here shall be the point of contact for any queries relating to data entered into this form about this mechanism) (CTRL+Click to select multiple values)	<div>Vladimir Stankovic</div> <div>Yuhui Chen</div> <div>Yves Crouzet</div> <div>Yves Deswarte</div> <div>Yves Roudier</div> <div>Zoe Andrews</div>
--	---

Figure 6: An example multi select

Multi Select Allowing Additions

This is very similar to the multi select type described above but allows the user to add items to the list of possible answers that can be selected by clicking on the "Add new item" link below and to the right of the list of options (e.g., Figure 7).

Categorisation (Select the categories that your mechanism falls into. If there are relevant categories not present in this list click on the "add new item" link to create an additional category) (CTRL+Click to select multiple values)	<div>Design-time</div> <div>Provides Fault Forecasting</div> <div>Provides Fault Prevention</div> <div>Provides Fault Removal</div> <div>Provides Fault Tolerance</div> <div>Resilience Architecture</div>
---	--

[Add new item]

Figure 7: An example multi select allowing additions

The “Add new item” link opens a sub form that allows the user to add the details of the new item. An example of this sub form is shown in Figure 8. Users are advised to employ meaningful names and provide a clear, generic description of the new item as these items may be used subsequently in other mechanism descriptions.

Figure 8: An example sub form for adding items to a multi select allowing additions list

Searchable Item

The searchable item entry type allows users to search the RKB for items to associate with a mechanism, including people, publications and existing mechanisms. This is used in cases where the option of selecting items from a list would be time consuming due to the huge number of possible answers. The user clicks on the “Add new item” link to reveal a search form. Existing items of this type can be deleted if they are no longer wanted by clicking on the red cross next to the summary (Figure 9).

Figure 9: An example searchable item

Users are currently recommended to use as specific a search term as possible to reduce search time. Correct items are selected from the search results and saved. A search may return many versions of a correct item (Section 3.2.3 explains why this happens); selection of any option is sufficient. If the sought item is not found, it is possible to add the required resource to the RKB. Clicking on the “Add new resource to RKB” link below the search results takes the user to a sub form similar to the one shown in Figure 8.

Figure 10 shows an example search form in which the link for adding new items to the RKB has been circled.

Select Diagrams
(If applicable, click on the "add new item" link to search for, and add, documents/web pages that contain a diagrammatic representation of your mechanism)

Please enter search term [Search]

Resources:
(CTRL+Click to select multiple values)

- Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures
- Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures

[Add new resource to RKB]

Save

Cancel

Figure 10: An example sub form for searching for items



Composite Item

Composite items allow users to enter data into the sub fields of an instance of resilience metadata, such as the metadata type, value and units, and then associate this resilience metadata instance with the mechanism. The user clicks on the “Add new item” link below and to the right of the question, leading to a sub form that allows data entry for the sub fields. A summary of the composite item will be displayed as a link in the main form. In the case of resilience metadata the information shown in this link is of the form <Metadata type> <Value> <Units>. Clicking on the link takes the user to the full details of the composite type in the triple browser (see Section 3.1.2). Two buttons also appear next to each composite entry link permitting editing or deletion of the entry. After editing a composite entry both the updated version and its original version will be associated with the mechanism description; the older version can be deleted by clicking on the cross next to it. Figure 11 shows an example of a composite entry, with the edit button circled.



Resilience Metadata

In this question you are asked to think about the effect your mechanism has on the resilience of a system. If you were to compare your mechanism to a different mechanism addressing a similar aim, what data would you use to choose which was fit for a specific purpose?

POFOD (Undetected) Prob(activating a related fault among the variants and the acceptance test) Probability  

[\[Add new item\]](#)

Figure 11: An example composite item

Check Boxes

Check boxes are used for selection from a structured group of options such as a hierarchy. In the case of the Res-Ex interface, check boxes allow the user to associate research interests from the ReSIST ontology on dependability and security with their mechanism. This approach is used because the ontology is highly structured. Figure 12 shows an excerpt from the check box representation.

A node in the tree structure is selected, by ticking the adjacent box. When ticked, this node, and all of its children, is highlighted in yellow to indicate the selection that has been made. Selecting a node automatically selects all of its children, for example if it is stated that a mechanism relates to “Dependability, High Confidence and Survivability” it also means that the mechanism relates to “Dependence”, “Trust”, “Attributes of Dependable Systems”, “Availability”, etc., as is shown in Figure 12. If the user then decides that the mechanism actually relates to a more specific aspect of “Dependability, High Confidence and Survivability” such as “Trust” they should unselect the former and tick next to “Trust” instead. Note also that each node is a sub-topic of its parent node; therefore a mechanism is related to the parents of the selected nodes as well. Care is needed, when users select related research areas for their mechanisms, to ensure that all of the research areas that are selected (either manually, or automatically in the case of children of selected nodes) are indeed directly related to the mechanism.

☐ Dependability And Security, Trustworthiness

Two somewhat overlapping concepts, with dependability being an integrating concept that encompasses the attributes: availability, reliability, safety, integrity and maintainability, while security encompasses confidentiality as well as integrity and availability.

☒ Dependability, High Confidence, Survivability

The original definition of dependability is: the ability to deliver service that can justifiably be trusted. The alternate definition, that provides the criterion for deciding if the service is dependable, is: the dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable.

☒ Dependence

The dependence of system A on system B represents the extent to which system A's dependability is (or would be) affected by that of System B.

☒ Trust

Accepted dependence - where the dependence of a user on a given system represents the extent to which the user's dependability is (or would be) affected by that of the system. (The acceptance of this state of affairs by the user may be willing or unwilling, and careful or even unthinking.)

☒ Attribute Of Dependable Systems

The dependability properties that are expected from a system, and in terms of which a system's dependability can be assessed with respect to the threats and the means to oppose these threats.

☒ Availability

1) Readiness for correct service; 2) Measure of the delivery of correct service with respect to the alternation of correct and incorrect service.

☒ Integrity

The absence of improper system alterations.

☒ Maintainability

1) Ability to undergo repairs and modifications; 2) Measure of the continuous delivery of incorrect service; 3) Measure of the time to restoration from the last failure occurrence.

Figure 12: An example check boxes item showing part of the dependability and security ontology

3.2.3 Common Problems

Several common problems were identified when the first set of mechanisms were entered into the RKB via the interface by the users. We describe them here in order to clarify the underlying issues and to help prevent future users of the Res-Ex interface from making the same mistakes.

Confusion about the scope of the mechanism description

When creating a resilience mechanism description, it is important to be clear and consistent about what is to be described, and at what level of specificity. For example recovery blocks could be described as a concept or general technique for providing

backward error recovery, however a specific implementation of it as a mechanism (such as Recovery Blocks/1/1, see Section 2.11) is also a valid and useful mechanism description. It is important that the reader knows exactly what is being described and that this remains constant throughout the description. A common mistake is to try to describe both the generic technique and a specific implementation of it within the same resilience mechanism description.

Resilience metadata types and values

When adding a new item of metadata, the user is asked for the type of the metadata. This is intended to represent a specific metric/property (such as mean time between failures, workload, etc.). New items can be added to the set of metadata types in the RKB if the required metric/property is not already present. The value box is meant to be a value/formula for the metric/property that applies to the specific mechanism being described. A common mistake is to use a high level item for the metadata type, such as reliability, and then describe the metric of reliability being used in the value box. This is not the intended use of these two fields. The specific metric/property should be entered into the metadata type and the value field should be the value of, or equation for calculating, this metric/property specific to the mechanism (or should be left blank if the value is obtained dynamically when the mechanism is in use).

When adding new metrics/properties to the list of metadata types, it is important to be as precise as possible in the definition to reduce the risk of ambiguity in multiple uses of the type. Likewise when using an existing metric/property, it is necessary to check that it is exactly what is required by reviewing the definition (revealed by hovering the mouse over the metadata type).

Multiple replicated items in search results

When using search forms in the Res-Ex interface to query the RKB, a query may generate a list of results that includes several different versions, or replicas, of the same item. The data in the RKB is derived from many different sources, each with its own representation of people, publications, etc. It is non-trivial to know that, for example, “J Smith” from one source is the same individual as “Jim Smith” from a different source. Ongoing work (at Southampton) aims to find means of discovering such correlations automatically and collating data from different sources in a reliable way without multiple representations of the same item. For the time being, if such a situation occurs, the user can select any one of the valid options.

Not noticing the “Add new item” links

When answering questions with a “multi select allowing additions” entry type (see Section 3.2.2), a number of people did not notice the “Add new item” link. This meant that they either selected items that were already in the list but were not exactly what they wanted, or omitted to answer the question even though it was relevant to their mechanism. Users are advised to check for the “Add new item” option when choosing options from a multi select list.

Incorrect usage of the description field when adding new items

When adding a new item to a multi select allowing additions entry type field (see Section 3.2.2), users are asked to provide a name and a description. Items that are added in this way are available for any mechanisms to use. Thus, it is important that the description section should provide a clear definition of the item added. A common mistake is to use the description field of the item to describe how the item relates to the mechanism for which it was added.

Relationship to working group comments

Step 2 of the questionnaire asks users to associate their mechanism with the new ReSIST Work Package 2 Working Groups (Assessability, Diversity, Evolvability, Usability). The subsequent four questions request the user to comment, if applicable, on the relationship between the mechanism and these working groups. It was intended that these four questions only be answered for the working groups selected in the prior multi select. For example, if a user records that their mechanism relates to diversity and evolvability, comments on the relationship to each of these working groups should be provided in the appropriate text boxes. A common mistake is to enter comments in these text areas even when the corresponding working groups had not been selected.

3.3 Mechanism Description Fields

In this section, we describe the information about resilience mechanisms that is stored in the RKB. The guidance given in this section will assist the reader in interpreting the information in the RKB. It should also help a contributor to enter meaningful information about their own mechanism into the RKB. Specific guidance is given for every question in the questionnaire along with the entry type for providing such data through the Res-Ex interface. These entry types correspond to those described in Section 3.2.

3.3.1 Overview

The fields described in this section relate to the questions in step 1 of the online interface to the RKB for entering resilience-explicit descriptions of mechanisms. This step of the questionnaire provides a general overview of the mechanism.

Field	Entry Type	Guidance
Name of the Resilience Mechanism	Text field	A title for the mechanism. This title should adequately identify the mechanism.
Submitted By	Multi select	This records the contributor of the mechanism. The person(s) identified here shall be the point of contact for any queries relating to data entered into the interface about this mechanism.
Author of Mechanism	Searchable item	This records the authors of the mechanism. These people should have a good understanding of the mechanism and may be the same as those identified in the previous question.
Associated Projects	Searchable item	This records projects associated with this mechanism. Possible associations include projects that funded research on the mechanism, address similar aims or use similar techniques.

Mechanism Objectives	Text area	Summary of the purpose of the mechanism in a sentence or two.
Detailed Description	Text area	A detailed description of the mechanism, which should be detailed enough for the reader to be able to re-create the mechanism. Either type the detailed description into this text box or reference a paper containing such a description in the next field.
Detailed Description Publication	Searchable item	If applicable (see above), this record refers to papers providing a detailed description of the mechanism.

3.3.2 Classification

The fields described in this section relate to the questions in step 2 of the online Res-Ex interface to the RKB. This section provides classification of the mechanism in terms of the structure and usage of the mechanism as well as the benefits of it.

Field	Entry Type	Guidance
Related Working Groups (original)	Multi select	Records the original ReSIST Work Package 2 Working Group(s) (Architectures, Algorithms, Socio-Technical, Verification, Evaluation) that are related to this mechanism.
Related Working Groups (new)	Multi select	Records the new ReSIST Work Package 2 Working Group(s) (Assessability, Diversity, Evolvability, Usability) that are related to this mechanism. For each Working Group selected here, the appropriate comment boxes, from the next four fields, should be completed.
Relation to Assessability	Text area	If assessability is selected in "Related Working Groups (new)", this field should provide a comment on how this mechanism relates to assessability.
Relation to Diversity	Text area	If diversity is selected in "Related Working Groups (new)", this field should provide a comment on how this mechanism relates to diversity.
Relation to Evolvability	Text area	If evolvability is selected in "Related Working Groups (new)", this field should provide a comment on how this mechanism relates to evolvability.
Relation to Usability	Text area	If usability is selected in "Related Working Groups (new)", this field should provide a comment on how this mechanism relates to usability.
Categorisation	Multi select allowing additions	Provides categorisation of the mechanism for a number of different classification schemes. This currently includes: whether the mechanism is used at run-time or design-time; whether it is a tool, architecture, process, etc.; the means for attaining resilience – fault prevention, fault tolerance, fault removal or fault forecasting. Additional categories or classification schemes can be added to this list as appropriate.
Application Domains	Multi select	An indication of the application domain(s) the mechanism has either been used in or may be suitable for use in. Application areas include finance, manufacturing, aerospace, etc. and are taken from the ACM classification system.

3.3.3 Further Details

The fields described in this section relate to the questions in step 3 of the online Res-Ex interface to the RKB. In this section, further details such as the structure and variants of the mechanism are provided.

Field	Entry Type	Guidance
Related Concepts	Multi select allowing additions	This field provides key words for the mechanism. Important concepts relating to the mechanism should be listed here. Concepts can be added to this list as appropriate and a description of the concept should be provided when doing so.
Main Components	Multi select allowing additions	If the mechanism is component-based, e.g., an architecture, this field records the main components constituting the mechanism.
Mechanism Variants	Text area	Describes any variants of the mechanism (if any variants exist).
Related Resilience Mechanisms	Multi select allowing additions	This field provides a “see also” function, which refers the reader to mechanisms related to this one. Possible associations include mechanisms that address similar aims or use similar techniques. Mechanisms can be added to the list offered. This has the side effect of creating a stub for this new mechanism in the list of mechanism descriptions.

3.3.4 Prerequisites

The fields described in this section relate to the questions in step 4 of the online Res-Ex interface to the RKB. This section provides details of any prerequisites, such as knowledge and infrastructure, for using the mechanism.

Field	Entry Type	Guidance
Application Technologies	Multi select allowing additions	Lists the types of system to which the mechanism can be applied. For example, triple modular redundancy can only be applied successfully to technologies in which design failures do not play a significant part, e.g., hardware technology. New technologies can be added to this list.
Knowledge Requirements	Multi select allowing additions	Lists any areas of knowledge that may be needed prior to using this mechanism. For example, a verification tool may require some knowledge of the underlying formalism. New knowledge areas can be added to this list.
Infrastructure Requirements	Multi select allowing additions	Lists any particular infrastructure elements the use of this mechanism requires. For example, a tool may only run on Windows XP and may need a Java runtime environment. New infrastructure elements can be added to this list.
Other Prerequisites	Text area	Describes any other prerequisites there may be for using this mechanism that are not covered by the previous fields.

3.3.5 Resilience Metadata

The fields described in this section relate to the questions in step 5 of the online Res-Ex interface to the RKB. This section takes a detailed look at how the mechanism improves the resilience of a system. In this section, domain specific metrics are associated with a mechanism. New metrics may be defined as required.

Field	Entry Type	Guidance
Failure Modes	Multi select	Lists the ways in which this mechanism can fail to function as intended. If the mechanisms were considered as a black box, these are the kinds of failures one might observe from it. The failure modes are all taken from the ReSIST ontology on resilient, survivable and dependable systems.
Threats Addressed	Multi select	Lists the threats to resilience that this mechanism aims to address, i.e., the faults it aims to remove, the errors it aims to compensate for and the failures it aims to prevent. The threats

		are all taken from the ReSIST ontology on resilient, survivable and dependable systems.
Resilience Metadata	Composite	<p>This question provides information about the effect this mechanism has on the resilience of a system. The aim is that the additional domain specific metadata described here could be used to compare several different mechanisms that address a similar aim and support the selection of one that is most fit for a specific purpose. New metrics can be defined and mechanism-specific values can be assigned to metrics.</p> <p>Details of the fields of this composite metadata are given in the table below.</p>

Resilience Metadata Fields

This section describes the fields of the composite field, Resilience Metadata, which is used for describing items of resilience metadata specific to the mechanism.

Field	Entry Type	Guidance
Metadata Type	Multi select allowing additions	The metric/field for the metadata. A precise description is given for each metric that allows accurate interpretation and comparison of the values assigned to this metric. For example, the metric "Mean Time Between Failures" could be described to provide the mathematical arithmetic mean amount of time between failures of the system. Other metrics may instead require some textual description of less formal properties. When entering metadata and choosing a metric/field for it, take care to make sure that the definition of the metric is the same as your understanding of it as it will be expected that values of the same metadata type can be meaningfully compared. New resilience metadata metrics/fields can be added to the list. (Note that, for technical reasons, the addition of new metadata metrics/fields to the list must be done with Firefox web browser, at least version 2.0).
Value	Text area	A value or equation expressing the value of this metric/field for this mechanism. If the value is acquired dynamically at run-time, this may be stated here, or it may be the case that this field is left empty.
Units	Text field	States the units in which the value is expressed, if applicable. For example a mean time between failures may be expressed in seconds, minutes, hours, etc.
Acquisition Method	Multi select	The method by which the metadata can be acquired. In particular if the metadata is acquired dynamically at run time, it should be stated here whether such metadata is published by some part of the mechanism itself, or whether a third party is required to observe the behaviour of the mechanism and infer the metadata from their observations. It also states whether this metadata can (or needs to) be calculated from other metadata.

3.3.6 Supporting Documents, if applicable

The fields described in this section relate to the questions in step 6 of the online Res-Ex interface to the RKB. This section provides references to any supporting documents (including web pages) that provide additional useful information about the mechanism to the reader.

Field	Entry Type	Guidance
Formal Description	Searchable item	Provides references to documents/web pages that contain a formal description of this mechanism, if such publications exist.
Ontology	Searchable item	Provides references to documents/web pages that contain an ontology of the main concepts relating to this mechanism, if such publications exist.
Diagrams	Searchable item	Provides references to documents/web pages that contain diagrammatic representations of this mechanism, if such publications exist.
Examples	Searchable item	Provides references to documents/web pages that contain examples of the use of this mechanism, if such publications exist.
FAQ	Searchable item	Provides references to documents/web pages that contain frequently asked questions and answers for this mechanism, if such publications exist.
Other Related Publications	Searchable item	Provides references to any other documents/web pages that contain relevant information about this mechanism.

3.3.7 Research Areas

The field described in this section relates to the question in step 7 of the online Res-Ex interface to the RKB. This section of the interface allows the user to indicate the research interests from the ReSIST ontology that relate to their mechanism.

Field	Entry Type	Guidance
Research Interests	Check boxes	Records research interests that are related to this mechanism. All research interests are taken from the ReSIST ontology on resilient, survivable and dependable systems.

4 RKB: Overview and Res-Ex Extensions

The goal of this area of ReSIST's work is to develop a resource that can help the community gain the most value from its knowledge assets. The Resilience Knowledge Base (RKB) utilises powerful Semantic Web technologies to store detailed descriptions of resources (people, projects, publications, courses), and offers many benefits over and above the results of applying conventional web-based search engines, enabling rich annotation and the synthesis of data from disparate sources. For example, through the application of ontological mapping, researchers may access information which is relevant to their query terms, but which has originated from sources or been described in vocabularies with which they are not familiar.

This section provides an overview of RKB technologies and content (Sections 4.1-4.2) and then goes on to describe the extensions created for supporting metadata oriented descriptions of resilience mechanisms, the Res-Ex ontology (Section 4.3).

4.1 RKB Technologies

The Resilience Knowledge Base is built on several key Semantic Web technologies. The underlying data model is that of Resource Description Framework (RDF) [Klyne et al., 2004], and we use the open source *3store* repository [Harris et al., 2003] to provide large

scale storage and ontological inference while also facilitating query access over the information held.

RDF allows knowledge to be expressed in ‘triples’ in the form (<subject> <predicate> <object>). Each triple prescribes that some information resource or concept (the subject) has a particular property or relationship, defined by the predicate, to another given resource or value (the object). This structure is designed to permit formal logical reasoning, and is a natural way to describe the vast majority of data processed by computers. In RDF, the subject and object are each identified by a Universal Resource Identifier (URI), just as used in a link on a Web page. The predicates are also identified by URIs, enabling anyone to define a new concept or relation by defining a URI for it somewhere on the Web.

Ontology languages, such the Web Ontology Language (OWL) [Patel-Schneider et al., 2004], permit the description of ‘vocabularies’ defining further structure for information represented in RDF documents. Resources can be identified as instances of typed classes, which identify a particular concept and prescribe the properties (predicates) expected of instances of that class. In addition, metadata can be defined prescribing the manner in which properties and classes are intended to be used within the RDF data, indicating the valid range and domain of a property or the data type in the case of literal values. Sub-class relationships can be defined for classes and predicates, forming hierarchical structures, and support is provided for grouping resources and values together.

Other useful capabilities often include notions of ‘same-as’ and ‘distinct-from’, enabling concepts in different ontologies to be mapped together - an important advancement for creating a unified, interoperable information resource. These facilities allow OWL ontologies to make use of and extend parts of other OWL ontologies, in addition to mapping between common concepts and instances. This makes data expressed in an ontologically mediated manner both reusable and repurposeable by computer programs.

4.2 RKB Content

Within the RKB, there are in excess of 50 million RDF triples, corresponding to several ontologies:

- The Advanced Knowledge Technologies (AKT) ontology details concepts from within the academic domain, in particular details of people, projects, papers and organisations.
- The ReSIST Ontology, based on the “ALRL” taxonomy [Avizienis et al., 2004], covers concepts within the fields of research in resilient, survivable and dependable systems.
- The ReSIST Courseware Ontology represents the various educational courses and resources within the ReSIST project.
- The ACM Classification Ontology describes the ACM Computing Classification System for categorising papers.

- The ReSIST Resilience Mechanisms (Res-Ex) Ontology represents metadata-oriented descriptions of resilience mechanisms. See Section 4.3 for more details.

All of the above ontologies are available online at <http://resist.ecs.soton.ac.uk/ontologies/>.

Wherever possible ontological concepts are integrated and reused, for example the use of various AKT classes such as Person and Publication within the Courseware and Res-Ex ontologies. Furthermore, within the RKB, an ontological mapping between ACM categories and concepts from the ReSIST Ontology is provided to aid data reuse and interoperation.

4.3 Res-Ex Ontology

The Res-Ex interface to the RKB, described in Section 3, provides a method for entering data into the underlying Res-Ex ontology and is designed in such a way that the technicalities of this ontology are invisible to the user. It is nevertheless important that such an ontology exists. The ontology, written in OWL, defines the semantics of and relationships between resilience-related concepts. If all of the mechanisms are described in a standard way with respect to the ontology, meaningful comparisons can be made between mechanisms as the semantics of such descriptions will be well defined. Describing mechanisms against a standard ontology also allows machine interpretation and manipulation of the metadata. This is very important for dynamic reconfiguration, a target application of the resilience-explicit computing approach, where decisions governing selection and configuration of mechanisms may be, at least in part, automated.

When designing the Res-Ex interface and ontology, it was important to take into account the purpose it needed to fulfil. Several open “brainstorming” sessions were held during the ontology design process to consider the competency questions that the ontology should support. A list of competency questions derived from these sessions can be found in Appendix C.

The questions to be answered about resilience mechanisms were drafted and reviewed prior to implementing the Res-Ex interface and designing the underlying ontology. Version 1 of the questionnaire was trialled in the first Res-Ex workshop, when initial attempts were made to describe resilience mechanisms. Using feedback from this experience, the questionnaire was reviewed and version 2 of the questionnaire was created. Both versions 1 and 2 of the questionnaire are available on the ReSIST wiki. The questions found in the current version of the Res-Ex interface are very closely related to version 2 of the questionnaire.

When designing the questionnaire and ontology, some insight was taken from other ontologies and similar approaches such as: software design patterns [Coplien et al., 1995]; dependability and security patterns from the SERENITY project⁵; the AKT ontologies⁶; the ReSIST ontologies⁷ and courseware interface⁸.

⁵ <http://www.serenity-project.org/>

⁶ <http://www.aktors.org/ontology/portal>

⁷ <http://resist.ecs.soton.ac.uk/ontologies/>

⁸ <http://resist.ecs.soton.ac.uk/courseware/>

The full Res-Ex ontology is written in the web ontology language OWL and can be found at <http://resist.ecs.soton.ac.uk/ontologies/resilience-mechanisms.owl>. In this report, we just describe the essence of it and how it fits in with the existing RKB and Res-On ontology. The main class in the ontology is *Resilience Mechanism*, “a mechanism designed to improve the resilience of a computer based system”. There are numerous properties that are defined for the *Resilience Mechanism* class, some of them are DataType properties (associate textual or numerical data with an instance of the *Resilience Mechanism* class) and some are Object properties (associate instances of other classes with an instance of this class). Such properties include for example: *mechanism-objectives*, a DataType property which associates a *Resilience Mechanism* with a string of text summarising the objectives of the mechanism; and *has-related-mechanism*, an Object property which relates two distinct, but related in some way, instances of *Resilience Mechanism*. Whilst the *Resilience Mechanism* class is the main class in the Res-Ex ontology, there are also some other classes defined in the ontology, such as *Knowledge*, “a body of knowledge” and *Resilience Metadata*, “a piece of resilience metadata about a mechanism”. Properties are also defined for some of these other classes.

The Res-Ex ontology makes use of existing concepts, properties and instances in the RKB wherever possible, in fact the main class, *Resilience Mechanism*, is a sub class of the *AKT Technology* class. As such, all of the *AKT Technology* properties are inherited by the *Resilience Mechanism* class, for example *has-author*, which relates a piece of technology to a person that was involved in the creation of it. The Res-Ex ontology also uses the *AKT* classes for people, publications, projects and technologies as the object of *Resilience Mechanism* properties such as *has-associated-project*. This means, for example, that when a user wishes to associate a project with their mechanism, he/she can select any of the projects already in the RKB, which includes all of the CORDIS⁹ data on past and current projects. The Res-Ex ontology is also integrated with the ReSIST ontologies. The ACM classification ontology is used to provide a list of application domains with which mechanisms can be associated. The ReSIST ontology on resilient, survivable and dependable systems is used to describe research interests and threats addressed by a *Resilience Mechanism* as well as to classify the failure modes of them.

All of the questions on the main form in the Res-Ex interface relate to properties of the *Resilience Mechanism* class (with some of these properties being inherited from the *AKT Technology* class). For example the *mechanism-objectives* property relates to the “Mechanism Objectives” question found in the ‘overview’ section of the interface (see Section 3.3.1) and the *has-related-mechanism* property relates to the “Related Resilience Mechanisms” question in the ‘further details’ section of the interface (see Section 3.3.3).

The properties of the other classes in the Res-Ex ontology, such as the *Knowledge* and *Resilience Metadata* classes mentioned earlier, relate to questions found in the sub forms for adding new instances of these classes. For example, the *Resilience Metadata* class has a property *has-unit*, which relates to the “Units” question in the resilience metadata sub form (see Section 3.3.5).

⁹ <http://cordis.europa.eu/en/home.html>

In the future, we would like to extend the Res-Ex ontology to formally define relationships between the metadata metrics with the intention of promoting comparison between mechanisms. At this early stage, it is hard to anticipate the metrics that may need to be included in this ontology. Thus, we are taking a bottom-up approach and allowing the users to define the metrics that they require as they need them. Once a reasonable number of metrics have been entered into the interface, we can investigate the relationships between them and develop ontologies to represent such relationships.

5 Related Work

Our work on support for resilience-explicit computing has concentrated on providing a means of recording metadata-based descriptions of resilience mechanisms with the intention that the descriptions can be used to assist in the selection and configuration of mechanisms at design-time and run-time. Looking forward to run-time exploitation of metadata, there are several technologies supporting dynamic selection of components and services. In this section, we identify relevant existing work and place Res-Ex computing in this context, particularly noting those technologies in which metadata already plays a role. We see these as technologies that may be able to utilise metadata-oriented descriptions of resilience mechanisms.

5.1 Multi-Agent Systems

Matchmaking is the process of identifying a suitable service provider for a service requester using a middle agent. Providers of services advertise their service to middle agents (which store them), requesters of services address their requests to middle agents which then match the requests against the stored advertised services. Different matchmaking techniques exist based on different query and deductive languages. We can cite the Agent-Based Software Integration (ABSI) facilitator, one of the earliest matchmakers [Genesereth et al., 1993], where capabilities are described through the Knowledge Query Manipulation Language (KQML) and Knowledge Interchange Format (KIF). InfoSleuth [Bayardo et al., 1997] makes use of a common ontology for dynamically matching service requests to available resources. Larks [Sycara et al., 1999] is an agent capability description language supporting specification of context, type, I/O variables and constraints, and ontological descriptions of some terms in the specification. Larks specifically addresses matchmaking among heterogeneous agents. Besides work based on service specification, we can mention [Luan 2004] in which, in addition to the specification and domain ontology, a history of interactions with the agents is maintained in order to enhance run-time matching. Candidate services are first selected through semantic service description matching. In a second step, the performance rating of each candidate (based on the history of interactions) with respect to the specific request is estimated in order to determine the best candidate.

Matchmaking allows run-time selection of a service provider based on the description of its service, and possibly on additional non-functional criteria such as performance rating. The Res-Ex approach clearly leverages matchmaking, since dynamic function allocation, or re-configuration at run-time is similar to matchmaking based on resilience metadata. Res-Ex computing goes also much further than that, since the use of metadata

not only covers functional aspects, but also a large range of non-functional criteria, and the scope of resiliency is not limited to re-configuration or dynamic selection of services, but may encompass classic fault-tolerance, backward recovery, granting of resources at run-time, or cooperative backup.

5.2 Web Services

The Universal Description, Discovery and Integration (UDDI) registry concept provides a potential service integration capability in Web services. However, UDDI currently lacks sufficient metadata to describe service capability and relationships, inevitably making it difficult to discover the most appropriate service. Some current research [Zhou et al., 2005] is aimed at developing enhanced UDDI to discover services or their relationships dynamically by adding new metadata. Ontology-driven knowledge organisation could also provide benefits to current UDDI [Oh et al., 2005]. Semantic service discovery approaches aiming to improve precision in the search phase have also been presented [Canfora et al., 2005]. The semantic description of a service represents the functional aspects of the service expressed in a logical language and by means of formal ontologies. In [Arpinar et al., 2004], a collection of ontology-driven Web Services composition techniques are presented for discovering and assembling individual Web Services into more complex processes. Moreover, Quality of Service (QoS) is a key factor in Web Services and the Web Services composition itself can be adapted mainly because of QoS requirements at run-time.

Research on Web Services QoS-based selection and composition could benefit from Res-Ex work because, for example, the use of metadata based descriptions of resilience mechanisms could allow resilience mechanism selection at run-time to satisfy reliability and availability QoS parameters.

5.3 GRID computing

Computational grids are infrastructures that provide access to shared computing resources for a great number of users involved in large-scale collaborations. In the LHC Computing Grid (LCG) and Enabling Grid for E-science (EGEE) projects, the Grid Laboratory Uniform Environment (GLUE) schema defines a common conceptual data model for Grid resource discovery and monitoring [Delgado Paris et al., 2005].

There are neither protocols nor standards in the Grid community for dealing with ontologies [Gutiérrez et al., 2007]. However, ontologies can be used in Grid for several purposes: for describing policies and sharing information, services and computing resources in virtual organization, and for describing formal and informal properties of Grid resources and services. The OntoGrid project¹⁰ aims at developing a reference Semantic Open Grid Service Architecture (S-OGSA) for the development of distributed applications that need to use explicit and distributed metadata. The Web Services Data Access and Integration Ontology realisation (WS-DAIOnt) defines the data access services that are needed for dealing with ontologies in Grid environments [Gutiérrez et al., 2006]. [Corcho et al., 2007] describes the approach for metadata management proposed in the context of the S-OGSA. [Kaoudi et al., 2007] analyses the problem of

¹⁰ www.ontogrid.net

resource discovery in the Semantic Grid, showing how to solve this by utilizing Atlas, a P2P system for the distributed storage and retrieval of RDF(S) data. Atlas is being used to realise the metadata service of S-OGSA in a fully distributed and scalable way.

Grid computing encompasses resource discovery and resource allocations at run-time; it covers issues related to semantics and performance. It is an ideal application area for resilience-explicit computing. Techniques such as S-OGSA explicitly use metadata at run-time and propose a middleware architecture supporting metadata to be displayed, and shared among the different Grid entities. Focus is given on service provisioning and access control. Research on building reliable and high available GRID services could benefit from Res-Ex work: for example, metadata based descriptions of resilience mechanisms could allow strategies for enhancing reliability of services in S-OGSA.

5.4 Dynamic reconfiguration

Run-time reconfiguration of services has been reported in Wapee [Kim et al., 2006], a specific middleware that supports dynamic reconfiguration in case of detected faults. A dedicated Fault-Manager detects faults and a Run-time Service Manager triggers reconfiguration once faults have been detected. A Monitoring Service provides real-time monitoring and feedback status of jobs submitted to services. The detection of faults and the choice of the replacement component are based on formal description of faults, of functionality of services and of context information (resource requirements). Three ontologies are defined: fault ontology for types of faults and their causes; service ontology for functionality and resource requirements of services; and recovery strategy ontology for fault resolution.

In the field of autonomic computing, a uniform representation and composition of autonomic elements has been proposed [White et al., 2004], encompassing the use of a service-oriented architecture supporting the interactions of these elements, preliminary design patterns and policies. Accord [Liu et al., 2004] is a programming framework for autonomic applications, supporting the use of rules to control the behaviour and interaction of autonomic components. Dynamic addition, deletion and replacement of components are supported, as well as changes to interactions. Self-Managed Cells (SMC) [Dulay et al., 2005] consist of (heterogeneous) hardware and software elements and management services integrated through a common publish/subscribe event bus. Managed components are monitored and decisions and actions are taken on the basis of provided policies. SMC elements have well-defined expected interfaces, limiting the possibility for new elements to join the system, especially if they have not been designed by the same team. The SMC scheme does not specifically address the use of metadata, even though elements are monitored, which nevertheless implies that metadata is collected about their behaviour.

Dynamic reconfiguration of services tends to achieve goals similar to those of resilience-explicit computing at run-time. The use of metadata is not always “explicit” in the different approaches, but the use of metadata is present, since monitoring of a component implies checking some specific behaviour/performance/quality of service, etc. The above techniques focus on run-time architectures and middleware. Design-time issues are not yet primarily considered. The Res-Ex approach clearly encompasses dynamic reconfiguration of services and autonomic computing aspects in general. Res-

Ex computing intends to have a larger scope: it covers not only run-time activities related to resilience in the large (not only limited to reconfigurations), but also design-time activities by supporting the choice of resilience techniques at design-time.

5.5 Component-Based Software: selecting components

A design-time automated process for selecting, evaluating and testing third party components is presented in [Maxville et al., 2003] based on both metadata and formal specification of the required component (interface and behaviour) and of its context of use. Metadata capture context information and specific criteria of the desired component. The specification is used first to select the possible components on the basis of their expected functionality, and second to derive tests for evaluating the selected components in the targeted environment. This process leads to a ranking of short-listed components according to criteria such as performance, security, or ease of integration. The Z language is used for the specifications. The specification and the metadata are captured with XML. The above selection process has further been extended with AI techniques for classifying components in order to take into account interdependent criteria [Maxville et al., 2004].

This technique allows selection at design-time of the appropriate component. The main criterion is functionality; the use of the tests allows further identification of the components on the basis of additional non-functional criteria (e.g., performance). This type of work is completely in line with the resilience-explicit approach which allows both design-time and run-time use of metadata in order to ensure adequate choice of the right component. Even though the resilience-explicit computing approach is suited for configuration at design-time, and re-configuration at run-time by selecting components or services, it also goes further by supporting the use of metadata in order to define resilience strategies (different from re-configuration strategies).

6 Evaluation and Future Work

As indicated in Section 1, the goal of work on Res-Ex computing in ReSIST is to encourage the community to gather metadata-based descriptions of resilience mechanisms that can assist design-time and run-time decision-making. Our approach in the initial phase of work has been to capture a broad set of first edition mechanism descriptions by means of a prototype interface to the RKB. We have described these developments in Sections 2, 3 and 4 above. In Section 2, we have also included the main observations of the mechanism providers on the description process and interface.

All of the mechanisms originally offered were successfully recorded. Several contributors noted that metadata-based description encouraged them to make a careful examination of the mechanism proposed, including analytic evaluation (in the case of cooperative backup, Section 2.1) and careful thought about the failure modes of the mechanism itself (stochastic dependability evaluation tool and NVP, Sections 2.9, 2.10). It is also worth noting that the approach exposes limitations of the discourse surrounding certain mechanisms (e.g., supervisory systems, Section 2.6).

The breadth of the range of mechanisms included in the first edition led to a variety of challenges addressed by the contributors as they exercised the interface, RKB and

ontology. Several limitations were observed in the existing interface, metadata and ontology descriptions and these are discussed below. A common theme has been the need to be clear about the scope of the mechanism under consideration. This was identified as an issue for the ModelWorks tool (Section 2.3), dynamic function allocation (Section 2.5), NVP, Recovery blocks and N-self-checking programming (Sections 2.10-2.12). The issue is addressed in current guidance to users, and the existence of the first edition mechanism descriptions is itself a considerable help in encouraging and guiding new descriptions.

Future work will be in two strands. First, we will begin to address the exploitation of the mechanism and metadata descriptions already recorded, by encouraging the development of challenge problems that show how a Res-Ex approach may be realised (Section 6.1). Second, we will extend the collection of mechanism descriptions (Section 6.2), and improve the descriptions already recorded, as well as maintaining and improving the facilities for mechanism entry and viewing (Sections 6.3 and 6.4).

6.1 Exploitation of Metadata and Mechanisms

One of the goals of work on Res-Ex computing is to provide techniques and tools for constructing systems that may reconfigure predictably in response to impairments. In Section 5, we have identified several areas of current research, which suggest that it is possible to apply Res-Ex computing principles to develop systems that achieve predictable dynamic resilience through run-time adaptation. Achieving such resilience requires that reconfiguration is triggered by trustworthy metadata and governed by resilience policies that are well understood.

A Res-Ex system that exploits metadata at run-time requires the following elements in addition to resilience mechanisms:

- **Trustworthy resilience metadata** conveying *functional* information (e.g. pre/postconditions, represented by logical formulae or informal descriptions) and *non-functional* information (e.g. availability, represented by structured values) relevant to resilience. Metadata should be semantically interoperable. For example, in selecting an alternative component for a fault-tolerant assembly such as NVP (Section 2.10), it is important that the metadata for each component, such as probabilities of failure on demand, are known to refer to the same concept. Given semantic interoperability, analysis tools can compare components.
- **Policies governing resilience:** The architect must be able to define application-specific resilience policies that implement resilience mechanisms. Policies should be capable of being analysed, again formally, in advance of deployment in order to confirm that they will achieve the resilience properties required by the application. This in turn implies that the system architecture and the resilience policy have to be expressed sufficiently formally to give confidence in the outcome of analyses about whether a particular adaptation is viable. Policies might be implemented as controller programs that reconfigure application architectures in response to impairments (faults, detected threats, reduction in quality of service). For example, a policy might replace a component when its measured reliability metadata (based on probability of failure on demand) is observed to have fallen below a threshold level, or deteriorating in excess of a prescribed rate. Here the implementation of the policy must be able to

access the relevant metadata and perform a programmed comparison in order to trigger a reconfiguration. The search for a suitable substitute might include machine-assisted reasoning over the potential replacement's logical preconditions in order to ensure that it may be used safely in the reconfigured application.

- **Reasoning and adaptation services:** Architectures supporting dynamic resilience must include computation, reasoning and adaptation services that are strong enough to reason over the metadata needed to implement the adaptation policies. These must be backed up with other services to perform component searches and enact adaptation with minimal disruption as described by the resilience policy. In the scenario, these services are used for selecting and adapting components without compromising continuity of service.

As an example of the exploitation of metadata, Figure 13 [Di Marzo Serugendo et al., 2007] shows the elements of a run-time environment supporting a Res-Ex approach. In the example application (a GPS system), metadata on component availability (shown as a simple number for the sake of brevity) are maintained in a registry of “known” components. The decision to replace a component is governed by a comparison against availability metadata in the resilience policy program. The suitability of a replacement depends on its availability metadata and a simple functional description.

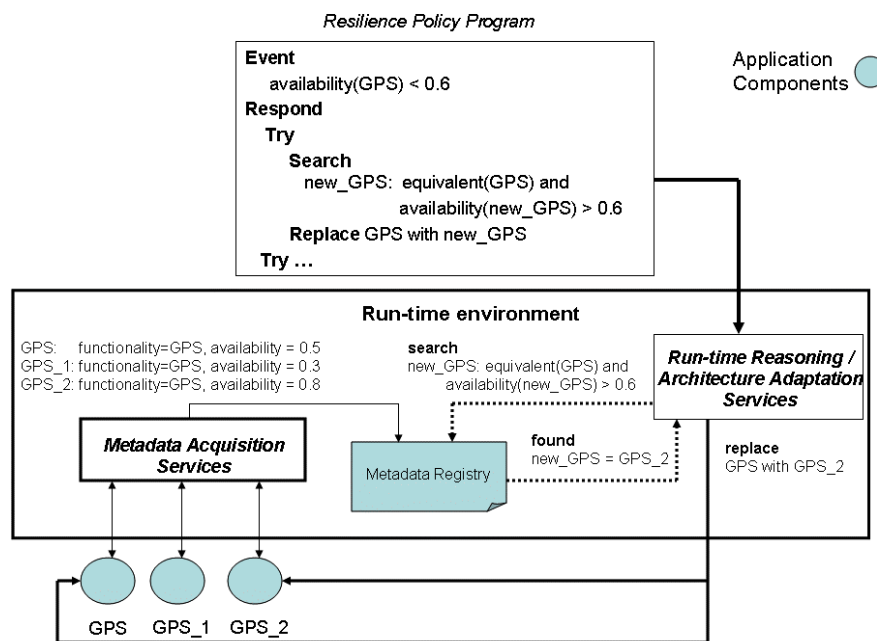


Figure 13: Run-time environment supporting dynamic resilience [Di Marzo Serugendo et al., 2007]

While there is ongoing research in each of the areas needed to realise a Res-Ex approach, there is need for advances of these along with *their successful integration*. In the next stage of ReSIST work on Res-Ex computing, we aim to develop a series of challenge problems identifying applications that require the run-time use of metadata to support dynamic reconfiguration for resilience. The development of the challenge problems is expected to deliver insight into the need for and role of metadata. Together, the problems will constitute a driver by which we hope to advance and integrate research.

By the end of the ReSIST project period, we will aim to have developed about three challenge problems. We aim to do this in collaboration with the Grand Challenges in Computing effort (http://www.ukcrc.org.uk/grand_challenges/current/index.cfm), especially GC6 on dependable systems evolution and GC2/4 on ubiquitous computing, both of which have developed strong international contributor bases. The challenge problems will serve as benchmarks for researchers and developers of tools supporting the design process or providing dynamic reconfiguration capabilities. Each will be based on an application (e.g., in transport, network communications or distributed control) and will invite contributors to develop and validate policies for dynamic resilience based on metadata. We expect the solutions proposed by contributors to include metadata-based descriptions of the resilience mechanisms deployed.

6.2 Second Edition Mechanisms

Work is planned to characterise further resilience mechanisms and so help explore the potential for Res-Ex outlined above. Below, we discuss the need for a second edition of resilience mechanisms, and then propose particular mechanisms.

6.2.1 Need for a Second Edition

It was not a goal of the first edition to be fully comprehensive. Our second edition content will aim to address gaps in the coverage provided by the sample of mechanisms described so far. We approach the identification of candidate second edition mechanisms in two ways. First by highlighting gaps in terms of the key characteristics in Table 1 (Section 2.13); and second by identifying resilience mechanisms familiar to participants in ReSIST.

ReSIST Partners. Given that the development of the RKB extensions and interface was a specialised task, the first edition mechanisms were contributed by members of the Res-Ex SIG: Newcastle (6 mechanisms), LAAS (1 mechanism), BUTE (3 mechanisms), QinetiQ (1 mechanism) and a ReSIST affiliate researcher (1 mechanism). In addition, there was active involvement via discussions and reviews from other partners. It is now desirable to obtain mechanism descriptions from a broader group, not least because this will expose the ontology and entry interface to users not directly involved in their development.

Mechanism Types. There are four tools/processes, and the remaining eight mechanisms are architectural. Three mechanisms are applicable at design/development time, and the remaining nine at run time. The described mechanisms apply to/within a wide variety of types of system, and can deliver a wide variety of direct benefits.

Coverage of Resilience Building and Scaling Technologies. There is a fairly good spread of mechanisms across the five original Resilience Building Technology areas identified with working groups in the ReSIST network. Most mechanisms are most closely associated with Algorithms and Architectures, two with Verification, one with Evaluation, and one with Socio-Technical. The coverage of Resilience Scaling Technologies is less broad: one is most closely associated with Evolvability, seven with Assessability, five with Diversity, and none with Usability.

6.2.2 Potential Second Edition Mechanisms

The following potential second edition mechanisms listed below have been identified as potential targets for description. They have been identified largely for pragmatic reasons, based on the knowledge of ReSIST partners evident in deliverables D12 *Resilience-Building Technologies: State of Knowledge* and D13 *From Resilience Building to Resilience Scaling: Directions*.

The following mechanisms were identified by detailed consideration of the research gaps and challenges identified in D13, focusing most on the ‘Current Approaches’ sections of each gap. This approach had varying degrees of success in terms of the numbers of potential second edition mechanisms found, most evidently in the relatively few Usability-related mechanisms. On the other hand, most of the identified mechanisms relate to Evolvability, to which, after Usability, the fewest first edition mechanisms relate.

In the listing below, we name the target mechanism and give the reference of the gap or challenge in D13.

Mechanisms relating to Evolvability:

Potential Mechanism	Gap/Challenge
Dynamic Coalitions (technique or modelling)	FE1: Evolution of Threats
Ad hoc routing in resilient ambient systems	FE2: Resilient Ambient Systems
DECOS modelling technique	FE3: Distributed System Models
SysML-based modelling	FE3: Distributed System Models
Intrusion tolerant middleware	FE4: Trustworthiness/Intrusion Tolerance in WANS
Intrusion tolerant architecture	FE4: Trustworthiness/Intrusion Tolerance in WANS
State machine replication	FE4: Trustworthiness/Intrusion Tolerance in WANS
Byzantine quorum systems	FE4: Trustworthiness/Intrusion Tolerance in WANS
Data protection (RAID/mirroring/replication/..)	FE5: Resilient Data Storage
Open component technology (OpenCom/Fractal)	FE7: Design for Adaptation: Framework and Programming Paradigms
Ad hoc wireless middleware (HIDENETS)	FE8: Adaptation
Virtualisation for dependable systems	FE11: Virtualisation

Mechanisms relating to Diversity:

Potential Mechanism	Gap/Challenge
Diversity against accidental and deliberate faults	FD1: Diversity for security, FD2: Large-scale Diversity for Intrusion Tolerance
Diversity and redundancy in security	FD1: Diversity for security, FD2: Large-scale Diversity for Intrusion Tolerance
Harmonizing diverse components by wrapping	FD3: Interoperability for Diversity

Mechanisms relating to Usability:

Potential Mechanism	Gap/Challenge
---------------------	---------------

User-Centered Design	<i>FU2: UCD & resilience engineering and development processes</i>
----------------------	--

Mechanisms relating to Assessability:

Potential Mechanism	Gap/Challenge
Testing of Distributed Mobile Systems	<i>FA11: Verification/Testing of Mobile Computing Systems</i>
Assume Guarantee Reasoning	<i>FA14: Compositional Reasoning</i>
Technique for Human Error Rate Prediction (THERP)	<i>FA17: Modelling Human Behavior</i>
Human Reliability Analysis	<i>FA17: Modelling Human Behavior</i>

Finally, the following potential second edition mechanisms were identified by considering D12. Particular focus was placed on the Socio-Technical and Verification parts, to counter the relatively small number of Usability (most closely related to Socio-Technical) and Assessability mechanisms listed above.

Mechanisms relating to D12 Socio-Technical:

Potential Mechanism	D12 Section
Cause-consequence analysis	<i>Socio 2: Evaluation and verification issues in resilience in socio-technical systems</i>
Human HAZOP / THEA / TRACer / Why Because Analysis	<i>Socio 2: Evaluation and verification issues in resilience in socio-technical systems</i>
Barrier based design of Interactive Systems	<i>Socio 1: Understanding the structure and organisation of socio-technical systems: representation and modelling</i>
IFADIS: Analysis of Dependable Interactive Systems	<i>Socio 1: Understanding the structure and organisation of socio-technical systems: representation and modelling</i>
Modelling Ambient and Mobile Systems	<i>Socio 2: Evaluation and verification issues in resilience in socio-technical systems</i>

Mechanisms relating to D12 Verification:

Potential Mechanism	D12 Section
TTP/C Bus Protocol	<i>Verif 1: Deductive Theorem Proving</i>
Abstract Interpretation	<i>Verif 3: Symbolic Execution and Abstract Interpretation</i>
Deductive Reasoning (theorem proving)	<i>Verif 1: Deductive Theorem Proving</i>

6.3 Entry Interface

The prototype Res-Ex interface was used successfully for the first edition resilience mechanisms. Issues raised during the study have been collected and recorded for addressing in future versions. These include the need to give the free mechanism description at an early stage (dynamic function allocation, Section 2.5).

Of the potential improvements identified in this first phase of use, several have already been implemented in a new version of the interface that is currently in

preparation. These include improved navigation to specific pages of the mechanism editing form and several layout enhancements. Some of the identified problems that have been resolved in the new interface relate to the use of browsers other than Firefox 2+ and errors in updating metadata descriptions following edit.

6.4 RKB Explorer Interface and Res-Ex Ontology

A prototype web-based interface has been developed to allow users to navigate the large quantities of information stored within the RKB. It is freely available for public use at <http://resist.ecs.soton.ac.uk/explorer/> and is shown in Figure 14.

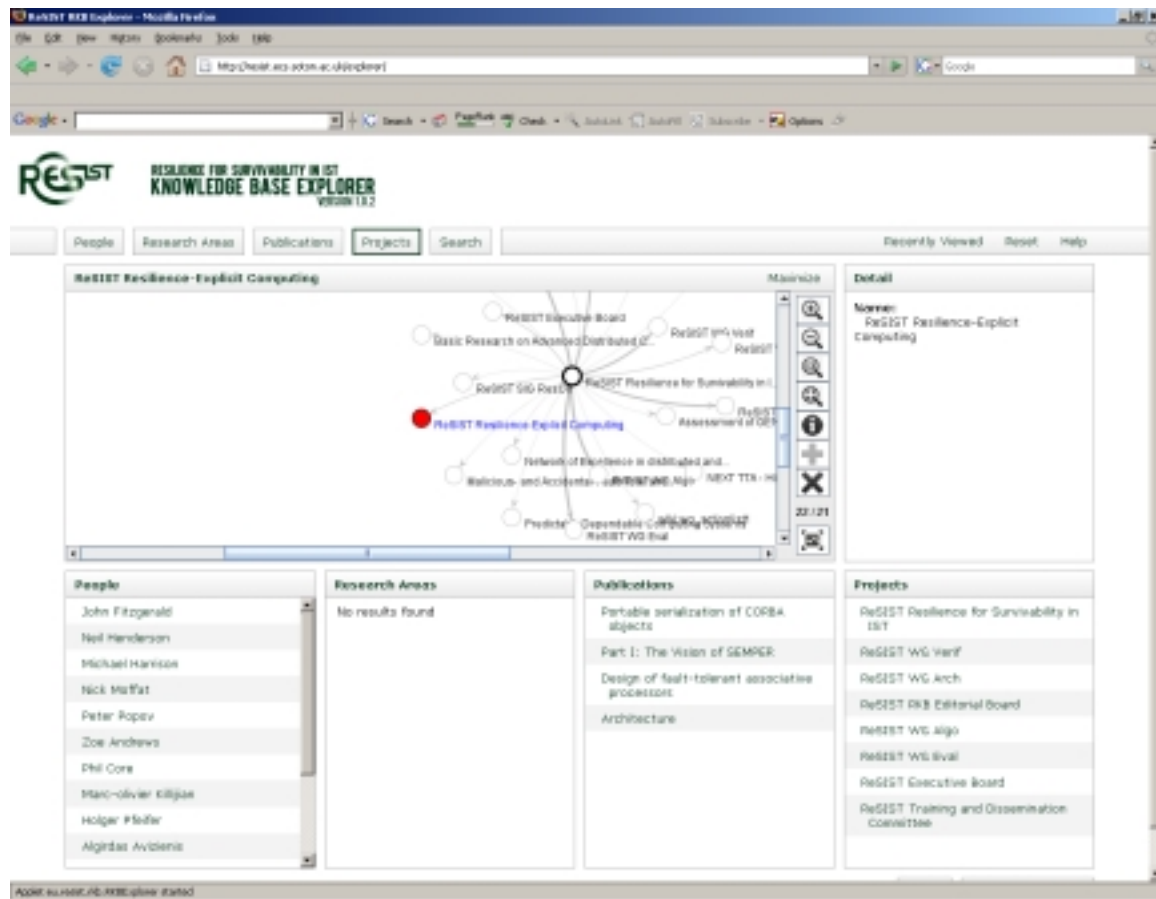


Figure 14: The Explorer interface to the RKB

Users may currently use this interface to search and browse through the data available based around the four themes of People, Research Topics, Publications and Projects. At any time, the top half of the interface window details an instance of one of these types of resource, while the lower half lists those resources of each type which are related to the currently selected item as determined automatically through semantic analyses.

Given the inclusion of semantic metadata detailing resilience-explicit computing mechanisms within the RKB, it is envisaged that a future release of the Explorer interface would present details and facilitate opportunistic discovery of these technologies. The Explorer would not only enable different mechanisms to be viewed as the focus of the

display, while showing other related mechanisms and associated papers, researchers, projects, and interest areas, but also, conversely, to highlight the relevance of these mechanisms while users are browsing other types of resource.

The Res-Ex ontology that underpins the RKB extensions to handle mechanism description (Section 4.3) proved adequate for describing the first edition mechanisms and this was achieved with considerable re-use of concepts from the AKT and ReSIST ontologies. In order to support decision-making, we would like to strengthen the Res-Ex ontology to handle relationships between metadata metrics and inheritance of metadata as suggested in the cases of consensus mechanisms (Section 2.2) and dynamic function allocation (Section 2.5). However, more metric and mechanism descriptions should be gathered to ensure that this is properly addressed. Concepts strong enough to handle descriptions of security and cryptography mechanisms are required as suggested by the re-encryption mixes mechanism (Section 2.4) and this is being addressed in the Resilience Ontology work within ReSIST.

6.5 Concluding Remarks

The description of resilience mechanisms is challenging but beneficial. Contributors have been forced to answer difficult questions about their mechanisms, in characterising as precisely as possible the effects that they have on overall system resilience. This challenge is perceived as one of the benefits of the approach, but it does suggest that gathering metadata-oriented descriptions will continue to require a high level of interaction between the maintainers of the knowledge base and the mechanism contributors. Bearing this in mind, we plan to continue to improve and update the RKB and Res-Ex interfaces.

We have concentrated so far on acquiring and recording mechanism descriptions, and not on their exploitation. Our proposed next phase of work aims to encourage the use of mechanism descriptions (via development of engaging challenge problems in metadata-based reconfiguration for resilience, inspired by some of the potential application technologies discussed in Section 5) and to encourage further contributions to the RKB as well as to its content. We will seek contributions from a wider range of researchers in ReSIST but, equally importantly, from beyond the network.

References

- [Arpinar et al., 2004] I.B. Arpinar, B. Aleman-Meza, R. Zhang, A. Maduko, Ontology-Driven Web Services Composition Platform, Proceedings of the 2004 IEEE International Conference on E-Commerce Technology (CEC 2004), pp. 146-152, 2004.
- [Avizienis 1985] A. Avizienis, The N-Version Approach to Fault-Tolerant Systems, IEEE Trans. on Software Engineering, Vol. SE- 11, No. 12, Dec. 1985, pp. 1491-1501.
- [Avizienis et al., 2004] 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, No. 1, pp. 11-33, January 2004.
- [Bayardo et al., 1997] R. J. Bayardo et al., The InfoSleuth Project, Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, 1997.
- [Cachin et al., 2000] C. Cachin, K. Kursawe, V. Shoup, Random oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography, Proceedings of the 19th Annual Symposium on Principles of Distributed Computing, Portland, Oregon, July 2000. ACM.
- [Canfora et al., 2005] G. Canfora, P. Corte, A. De Nigro, D. Desideri, M. Di Penta, R. Esposito, A. Falanga, G. Renna, R. Scognamiglio, F. Torelli, M. L. Villani, P. Zampognaro, The C-Cube Framework: Developing Autonomic Applications through Web Services, Proceedings of the Workshop on Design and Evolution of Autonomic Application Software (DEAS 2005), St. Louis, Missouri, ACM SIGSOFT SE-Notes, Vol. 30, No. 4, 2005.
- [Castro et al., 1999] M. Castro, and B. Liskov, Practical Byzantine fault tolerance, Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, Louisiana, USENIX Association, Berkeley, CA, pp. 173-186, 1999.
- [Coplien et al., 1995] J. O. Coplien, D. C. Schmidt (Eds.), Pattern Languages of Program Design, Addison-Wesley, Reading, MA, 1995.
- [Corcho et al., 2007] O. Corcho, P. Alper, P. Missier, S. Bechhofer, C. Goble, W. Xing. Metadata Management in S-OGSA, Proceedings of International Workshop on Collective Intelligence for Semantic and Knowledge Grid (CISKGrid 2007).
- [Courtes et al., 2006] L. Courtes, M.-O. Killijian, D. Powell, Storage Tradeoffs in a Collaborative Backup Service for Mobile Devices, Dependable Computing Conference, 2006. EDCC '06. Sixth European , vol., no., pp.129-138, Oct. 2006.
- [Courtès et al., 2007] "Dependability Evaluation of Cooperative Backup Strategies Edit resource", Ludovic Courtès, Ossama Hamouda, Mohamed Kaâniche, Marc- Olivier Killijian, David Powell to appear in Procs of the 13th IEEE Pacific Rim International

Symposium on Dependable Computing (PRDC'07), Melbourne, Victoria, Australia
December 17-19, 2007

[Dearden et al., 2000] Dearden, A., Harrison, M.D., Wright, P.C., Allocation of Function: Scenarios, Context and the Economics of Effort, *International Journal of Human-Computer Studies*, Vol. 52, No. 2, pp. 289-318, 2000.

[Delgado Paris et al., 2005] A. Delgado Paris, P. Mendez Lorenzo, F. Donno, A. Sciaba, S. Campana, R. Santinelli, LCG-2 User Guide, CERN-LCG-GDEIS-454439, 2005.

[Di Marzo Serugendo et al., 2007] G. Di Marzo Serugendo, J. S. Fitzgerald, A. Romanovsky, N. Guelfi, A Metadata-based Architectural Model for Dynamically Resilient Systems, in *Proc. 2007 ACM Symposium on Applied Computing*, Seoul, Korea, March 2007. ACM Press.

[Dulay et al., 2005] N. Dulay, E. Lupu, M. Sloman, J. Sventek, N. Badr, S. Heeps, Self-Managed Cells for Ubiquitous Systems, *Proceedings of the Third International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS 2005)*, St. Petersburg, Russia, September 25-27, 2005, Vol. 3685 of *Lecture Notes in Computer Science*, pp. 1-6. Springer, 2005.

[Genesereth et al., 1993] M. R. Genesereth and N. P. Singh, A Knowledge Sharing Approach to Software Interoperation, *Stanford Logic Group Report Logic-93-12*.

[Gutiérrez et al., 2006] M. Gutiérrez, A. Gómez-Pérez, Ó. Muñoz García, Ontology Access in Grids with WS-DAIOnt and the RDF(S) Realization, *Semantic Grid Workshop, GGF16*, Athens, 15th Feb 2006.

[Gutiérrez et al., 2007] M. Gutiérrez, A. Gómez-Pérez, Ideas for the Provision of Ontology Access in Grid Environments, *Knowledge and Data Management in GRIDs*, in D. Talia, A. Bilas, M. Dikaiakos (Eds.), *CoreGRID Series*, Springer US, 2007.

[Harris et al., 2003] S. Harris and N. Gibbins, 3Store: Efficient Bulk RDF Storage, *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems*, pp. 1-15, 2003.

[Horning et al., 1974] J.J. Horning, H.C. Lauer, P.M. Melliar-Smith, B. Randell, A Program Structure for Error Detection and Recovery, *Proceedings of the International Symposium on Operating Systems: Theoretical and Practical Aspects*, Rocquencourt, France, 23-25 April 1974, Gelenbe, E., Kaiser, C. (Eds.), *Lecture Notes in Computer Science* Vol. 16, pp. 171-187, Springer-Verlag 1974.

[IBM 2006] An architectural blueprint for autonomic computing. White paper. IBM. 2006. http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf.

[Inayat et al., 2006] P. D. Ezhilchelvan, Q.-A. Inayat, A Performance Study on the Signal-On-Fail Approach to Imposing Total Order in the Streets of Byzantium, International Conference on Dependable Systems and Networks, 2006 (DSN 2006), pp. 578–590, 2006.

[Kaoudi et al., 2007] Z. Kaoudi, I. Miliaraki, M. Magiridou, E. Liarou, S. Idreos, M. Koubarakis, Semantic Grid Resource Discovery in Atlas, in D. Talia, A. Bilas, M. Dikaiakos (Eds.), Knowledge and Data Management in GRIDs, CoreGRID Series, Springer US, 2007.

[Kim et al., 2006] Y. Kim, E. Kim, J. Kim, E. Song, I. Ko, Ontology Based Software Reconfiguration in a Ubiquitous Computing Environment, Proceedings of the 6th IEEE International Conference on Computer and Information Technology (CIT'06), 2006.

[Klyne et al., 2004] G. Klyne, J. J. Carroll, B. McBride, Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, 2004.
<http://www.w3.org/TR/rdf-concepts/>

[Knight et al., 1986] J. C. Knight and N. G. Leveson, An Empirical Study of Failure Probabilities in Multi-version Software, Proceedings of the 16th IEEE International Symp. Fault-Tolerant Computing, 1986, Computer Society Press, Los Alamitos, California, Order No. 703, pp. 165-170.

[Laprie et al., 1990] J.C. Laprie, J. Arlat, C. Beounes, K. Kanoun, Definition and Analysis of Hardware and Software Fault-Tolerant Architectures, IEEE Computer, Vol. 23, No. 7, July 1990, pp. 39-51.

[Liu et al., 2004] H. Liu, M. Parashar, S. Hariri, A Component-Based Programming Model for Autonomic Applications, in J. Kephart, M. Parashar (Eds.), International Conference on Autonomic Computing (ICAC'04), pp. 10-17, IEEE Computer Society, 2004.

[Luan 2004] X. Luan, Adaptive Middle Agent for Service Matching in the Semantic Web: A Quantitative Approach, PhD Thesis, University of Maryland, 2004.

[Majzik et al., 2007] I. Majzik, P. Domokos, M. Magyar, Tool-supported Dependability Evaluation of Redundant Architectures in Computer Based Control Systems, in E. Schnieder, G. Tarnai (eds.), Proceedings FORMS/FORMAT 2007, the 6th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems, 25-26 January 2007, Braunschweig, Germany, pp 342-352. GZVB, Braunschweig, Germany, 2007.

[Maxville et al., 2003] V. Maxville, C. P. Lam, J. Armarego, Selecting Components: A Process for Context-Driven Evaluation, Proceedings of the Tenth Asia-Pacific Software Engineering Conference (APSEC'03), 2003.

- [Maxville et al., 2004] V. Maxville, J. Armarego, C. P. Lam, Intelligent Component Selection, Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04), 2004.
- [Micskei et al., 2006] Z. Micskei, I. Majzik, F. Tam, Robustness Testing Techniques For High Availability Middleware Solutions, Proceedings of International Workshop on Engineering of Fault Tolerant Systems (EFTS 2006), Luxembourg, Luxembourg, June 12-14, 2006.
- [Oh et al., 2005] S.G. Oh, E.C. Lee, O. Park, M. Yi, Ontology-Driven Knowledge Organization - Enhancing UDDI Web Services in Korea Using Topic Maps, Proceedings of the American Society for Information Science & Technology, Charlotte, North Carolina, October 28 - November 2, 2005
- [Patel-Schneider et al., 2004] P.F. Patel-Schneider, P. Hayes, I. Horrocks, OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation, 2004.
<http://www.w3.org/TR/owl-semantics/>
- [Pease et al., 1980] M. Pease, R. Shostak, L. Lamport, Reaching Agreement in the Presence of Faults, J. ACM, Vol. 27, No. 2, Apr. 1980, pp. 228-234.
- [Ryan et al., 2006] P. Y. A. Ryan, S. A. Schneider, Pret a Voter with Re-encryption Mixes, Technical Report Number CS-TR: 956, School of Computing Science, Newcastle University, Apr 2006.
- [Sycara et al., 1999] K. Sycara, J. Lu, M. Klusch, S. Widoff, Proceedings of the 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace, March, 1999.
- [White et al., 2004] S. White, J. Hanson, I. Whalley, D. Chess, J. Kephart, An Architectural Approach to Autonomic Computing, in J. Kephart, M. Parashar (Eds.), International Conference on Autonomic Computing (ICAC'04), pp. 2-9, IEEE Computer Society, 2004.
- [Zhou et al., 2005] G. Zhou, J.J. Yu, Using Enhanced UDDI to Support Service Dynamic Discovery, Proceedings of Communications, Internet, and Information Technology (CIIT 2005), Cambridge, USA, 2005

Appendix A: Res-Ex Case Study in Overview

A.1 Introduction

A designer requires a system that tolerates one (sequential) hardware fault and/or one software fault. The designer has limited resources available and wishes to provide a cost effective solution. However, the system must also be as reliable as possible.

The designer knows about three fault-tolerant architectures that would tolerate the required faults:

- Recovery Blocks (RB/1/1)
- N-Version Programming (NVP/1/1)
- N-Self Checking Programming (NSCP/1/1)

The problem that was examined was which of these provided suitable cost and reliability levels.

A.2 Decision Making with Metadata

The following metadata was identified for the fault-tolerant architectures from a paper by Laprie and colleagues [Laprie et al., 1990]. In terms of costs and overheads the metadata shown in Table 2 applies. Metadata detailing the reliability aspects of the fault-tolerant architectures is given in Table 3.

Method	Total no. of variants required	Total no. of hardware components required	Other structural overheads	Operational time overheads (normal operation)	Operational time overheads (when errors occur)	Min (CFT/ CNFT)	Max (CFT/ CNFT)	Av (CFT/ CNFT)
No Fault Tolerance	1	1	None	None	N/A	1	1	1
RB/1/1	2	2	Acceptance test. Recovery cache	Acceptance test execution. Accesses to recovery cache	One variant and acceptance test execution	1.33	2.17	1.75
NVP/1/1	3	3	Voters	Vote execution. Input data consistency and variants execution synchronisation	Usually negligible	1.78	2.71	2.25
NSCP/1/1	4	4	Comparators and result switching	Comparison execution. Input data consistency and variants execution synchronisation	Possible result switching	2.24	3.77	3.01

Table 2: Overheads and cost metadata of fault-tolerant architectures

Method	P (Software failure on demand) (P (S)) = P (Detected software failure on demand) + P (Undetected software failure on demand)		Time dependent (approximation for short missions wrt mtbf)	
	P (Detected software failure on demand) (P (S, D))	P (Undetected software failure on demand) (P (S, U))	Reliability	P (undetected failure)
RB/1/1	$(P(I))^2 + P(ID) + P(2V)$	$P(RVD)$	$1 - (2 * (1-c) * \lambda_H + \lambda_S) * t$	$\lambda_{S,U} * t$
NVP/1/1	$3 * (P(I))^2 [1 - (2/3) * P(I)] + P(ID)$	$3 * P(2V) + P(3V) + P(RVD)$	$1 - \lambda_S * t$	$\lambda_{S,U} * t$
NSCP/1/1	$4 * (P(I))^2 * [1 - P(I) + (P(I))^2/4] + P(ID) + 4 * P(2V)$	$P(2V) + 4 * P(3V) + P(4V) + P(RVD)$	$1 - \lambda_S * t$	$\lambda_{S,U} * t$

Table 3: Reliability metadata of fault-tolerant architectures

The variables used in Table 3 above are defined as follows:

- P (I) is the probability of activating an independent fault in one of the variants
- P (ID) is the probability of activating an independent fault in the decider
- P (nV) is the probability of activating a related fault among *n* of the variants
- P (RVD) is the probability of activating a related fault among the variants and the decider
- λ_H is the failure rate of a hardware component
- λ_S is the total failure rate of the fault tolerant software (assuming the application's execution rate is λ this is equivalent to $P(S) * \lambda$)
- $\lambda_{S,U}$ is the undetected failure rate of the fault tolerant software (assuming the application's execution rate is λ this is equivalent to $P(S, U) * \lambda$)
- *c* is the hardware coverage factor of the recovery blocks architecture

There were also some additional properties of the fault-tolerant architectures, shown in Table 4, which may influence the decision between them.

Method	Additional properties		Fault-tolerance after a previous fault (and components disabled)	
	Hardware	Software	Hardware fault	Software fault
RB/1/1	Low error latency	None	Detection provided by local diagnosis	Tolerance of one independent fault
NVP/1/1	Detection of two or three faults	Detection of two or three independent faults	Detection	Detection of independent faults
NSCP/1/1	Tolerance of two hardware faults in the same self-checking component. Detection of two, three or four faults	Tolerance of two independent faults in the same self-checking component. Detection of two, three or four independent faults.	Detection	Detection of independent faults

Table 4: Additional properties metadata of fault-tolerant architectures

It was attempted to find actual data values to put into the formulae to calculate the reliability of the fault-tolerant architectures. However, acquiring such data proved to be very difficult and only some data values could be determined from previous work. Such

values are taken from Knight and Leveson's evaluation of N-Version Programming [Knight et al., 1986]. The remaining parameters were either derived from these, or suitable values were introduced (as this scenario simply aims to illustrate the decision making process it was thought that not too much time should be spent trying to get real data).

Using these values the following results were obtained for the reliability of the fault-tolerant architectures:

- *RB*: $R(60) = 0.984$
- *NVP*: $R(60) = 0.970$
- *NSCP*: $R(60) = 0.961$

The cost of the fault-tolerant architectures can be taken from the tables of metadata shown in Tables 1-3:

- *RB*: 1.33 to 2.17 (average of 1.75) CFT/ CNFT, plus 1 additional hardware component
- *NVP*: 1.78 to 2.71 (average of 2.25) CFT/ CNFT, plus 2 additional hardware components
- *NSCP*: 2.24 to 3.77 (average of 3.01) CFT/ CNFT, plus 3 additional hardware components

This showed that *RB* is the most reliable and also the cheapest to implement. However, if other metadata, such as the run-time overheads when errors occur, is also taken into account *NVP* or *NSCP* may be preferable.

Appendix B: Completeness of First Edition Mechanism Descriptions

This appendix provides an indication of the depth of each mechanism description in the RKB by stating, for each question, whether it has been completed (Y) or not (N) or is not applicable to the mechanism (N/A). The mechanisms are referred to by the section number in which they are described above, a reminder of which section refers to which mechanism is given below:

- 2.1 Cooperative Backup
- 2.2 Consensus Mechanisms
 - Mech refers to the top-level description of a consensus mechanism
 - BFT refers to Practical Byzantine Fault Tolerance
 - SOF refers to Signal-On-Fail based consensus protocol
 - Sintra refers to Secure Intrusion-Tolerant Replication Architecture
- 2.3 ModelWorks
- 2.4 Robust Re-Encryption Mixes
- 2.5 Dynamic Function Allocation
- 2.6 Supervisory Systems
- 2.7 Autonomic Computing Architecture
- 2.8 Robustness Testing
- 2.9 Model-based Stochastic Dependability Evaluation Tool
- 2.10 N-Version Programming/1/1
- 2.11 Recovery Blocks/1/1
- 2.12 N-Self-Checking Programming/1/1

Question	2.1	0				2.3	2.4	2.5	2.6	2.7	2.8	2.9	2.10	2.11	2.12
		Mech	BFT	SOF	Sintra										
Mechanism Name	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Submitted By	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Author of Mechanism	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y
Associated Projects	Y	N	N	N	Y	Y	N	N	Y	Y	N	Y	N	N	N
Mechanism Objectives	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Detailed Description	N/A	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N/A	Y	Y	Y
Detailed Description Publication	Y	Y	Y	Y	Y	N/A	Y	Y	Y	Y	Y	Y	Y	Y	Y

Related Working Groups (original)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Related Working Groups (new)	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Relation to Assessability	N/A	N/A	N/A	N/A	N/A	Y	Y	Y	Y	N/A	Y	Y	Y	Y	Y
Relation to Diversity	Y	Y	Y	Y	Y	N/A	N/A	Y	N/A	N/A	Y	N/A	Y	Y	Y
Relation to Evolvability	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Y	Y	Y	N/A	N/A	Y	Y	Y
Relation to Usability	N/A	N/A	N/A	N/A	N/A	N/A	N/A	Y	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Categorisation	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Application Domains	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Related Concepts	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Main Components	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Mechanism Variants	N	Y	N	N	N	N/A	N	Y	Y	Y	Y	Y	Y	Y	Y
Related Resilience Mechanisms	N	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y	Y	Y
Application Technologies	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
Knowledge Requirements	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Infrastructure Requirements	Y	N	N	Y	Y	Y	Y	N	Y	Y	Y	Y	N	N	N
Other Prerequisites	N	Y	Y	Y	Y	Y	Y	N	N	N	Y	Y	Y	Y	Y
Failure Modes	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Threats Addressed	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Resilience Metadata	Y	N	N	N	N	Y	Y	Y	N	Y	Y	Y	Y	Y	Y
Formal description	N	Y	N	N	N	N	Y	N	N	N	Y	Y	Y	Y	Y
Ontology	N	N	N	N	N	Y	N	N	N	N	N	Y	Y	Y	Y
Diagrams	N	N	N	N	N	N	Y	N	N	N	Y	N	Y	Y	Y
Examples	N	N	N	N	N	N	Y	N	N	N	Y	Y	Y	Y	Y
FAQ	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	N
Other Related Publications	Y	Y	N	N	Y	Y	Y	Y	N	Y	Y	N	Y	Y	Y
Research Interests	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y

Table 5: Indication of the completeness of the first edition mechanism descriptions

Appendix C: Competency Questions

This appendix provides the competency questions derived for the Res-Ex ontology:

- 1) What information exists about research area / mechanism x?
- 2) What mechanisms exist for research area x?
 - a) What is the competition?
 - b) What mechanisms exist that could help with my research?
- 3) What literature is there about research area / mechanism x?
- 4) Who's working in research area / on mechanism x?
- 5) Who's interested in research area / on mechanism x?
- 6) Who might be interested in a mechanism on x?
- 7) What benefit (to resilience) does mechanism x provide?
 - a) What does x do for availability / ... ?
- 8) What are the gaps (potential areas for new mechanisms) in research area x?
- 9) What are the frequently asked questions about a research area or mechanism and the corresponding answers?
- 10) What project(s) relate(s) to mechanism x?
- 11) What are the prerequisites for using mechanism x?
 - a) What are the environment assumptions?
 - b) What prior knowledge is required?
- 12) What genres of mechanisms exist?
 - a) What genre does mechanism x belong to?
 - b) What does this tell me?