Traceable Engineering of Fault-Tolerant SoSs

Traceable Engineering of Fault-Tolerant SoSs


Abstract

Systems of systems (SoSs) are characterised by a challenging combination of continuous evolution, emergent behaviour and distributed, autonomous, independent constituents. The development of SoSs that can tolerate faults and harmful events is hampered by these and other complexities. Currently there is little in the way of models or tools to help SoS developers to design fault-tolerant SoSs. In this paper we present a structured approach for capturing requirements for a fault-tolerant SoS and a fault modelling architectural framework (FMAF) that supports disciplined and reusable development of fault-tolerant architectures, as well as a traceable mapping of the fault-tolerant requirements into SoS architectural designs. Finally we present an example real-world SoS case study to demonstrate the application of our techniques.
Abstract

Systems of systems (SoSs) are characterised by a challenging combination of continuous evolution, emergent behaviour and distributed, autonomous, independent constituents. The development of SoSs that can tolerate faults and harmful events is hampered by these and other complexities. Currently there is little in the way of models or tools to help SoS developers to design fault-tolerant SoSs. In this paper we present a structured approach for capturing requirements for a fault-tolerant SoS and a fault modelling architectural framework (FMAF) that supports disciplined and reusable development of fault-tolerant architectures, as well as a traceable mapping of the fault-tolerant requirements into SoS architectural designs. Finally we present an example real-world SoS case study to demonstrate the application of our techniques.

About the authors

Zoe Andrews is a Research Associate on the COMPASS project. In particular, she is exploring ways of modelling and analysing faults in systems of systems. Zoe completed her PhD (supervised by Prof. John Fitzgerald) on "Continuous Probability Distributions in Model-Based Specification Languages" in 2012. This investigated ways in which stochastic reasoning could be combined with logical reasoning for the specification and analysis of fault-tolerant systems. Zoe also worked as an RA on the ReSIST network of excellence and was responsible for work on developing metadata-based descriptions of resilience mechanisms and providing formal support for decision making over such mechanisms.

Jon Holt is currently the Global Head of Systems Engineering for Atego, a leading independent provider of tools and capability for systems engineering, and was the founder-director of Brass Bullet Ltd, a systems engineering consultancy and training company for over 12 years, until it was acquired in 2009. Jon is also a Professor of Systems Engineering at the UK Defence Academy, where he is involved with teaching and research. Jon is active in the IET via their Professional Networks and the BCS as a member of the Learned Society. He is a Fellow of both the IET and the BCS and is a Chartered Engineer and Chartered IT Professional. Jon is an international award-winning author and public speaker in the field of applied systems engineering and research. He has authored nine books on systems engineering. His books cover the application of UML and SysML to systems engineering, process modelling, enterprise architectures and competency assessment. His main area of interest is the application of systems modelling to all aspects of system engineering and his areas of expertise include: UML and SysML for system engineering, process modelling, standards compliance, requirements engineering, life cycle modelling, enterprise architectures, architectural frameworks and competency assessment.

Claire Ingram earned her PhD at Newcastle University in 2011 by developing new software metrics to investigate whether early project data can be used to predict later change-proneness. After obtaining her PhD she contributed towards the DESTECS project, which developed new tools and techniques to support fault-tolerant embedded systems. Currently she works on the COMPASS project, a research consortium investigating the use of model-based techniques for developing and maintaining systems of systems.

Richard Payne obtained his PhD in 2012 at Newcastle University under the supervision of Prof. John Fitzgerald, titled Verifiable Resilience in Architectural Reconfiguration. As part of his PhD, Richard provided a basis for the formal verification of policies defined using a reconfiguration policy language (RPL) for the governance of resilient component-based systems. Richard worked as an RA on the Ministry of Defence funded SSEI project and was involved in the 'Interface Contracts for Architectural Specification and Assessment' sub task, investigating the use of contract-based interface specification in system of systems architectural models. Richard is now working on the COMPASS project, on the use of model-based techniques for developing and maintaining systems of systems, involved with work in architectural modelling, fault modelling and tool development.

Simon Perry holds Bachelor degrees from both the University of Leeds and the Open University. Since gaining his degree in Mathematics in 1986 he has spent over 25 years working in all aspects of software and systems engineering. Simon often speaks at systems engineering conferences and is the co-author of five books on systems engineering and related topics: 'SysML for Systems Engineering', 'Modelling Enterprise Architectures', 'Model-
Based Requirements Engineering' and 'Model-based Systems Engineering using SysML' published by the IET and 'A Pragmatic Guide to Competency' published by the BCS. Simon is a Principal Consultant for Atego Systems, providing consultancy, training and conducting research in the application of systems engineering. He works in industry, government and academia and has applied his work across many disciplines in a wide range of industries including defence, the nuclear industry, timber engineering, finance and train manufacture. He is a Member of the IET, INCOSE and is a part-time lecturer at the University of Warwick.

Alexander (Sascha) Romanovsky is a Professor in the Centre for Software and Reliability, Newcastle University. His main research interests are system dependability, fault tolerance, software architectures, exception handling, error recovery, system structuring and verification of fault tolerance. He received a PhD degree in Computer Science from St. Petersburg State Technical University and has worked as a visiting researcher at ABB Ltd Computer Architecture Lab Research Center, Switzerland and at Istituto di Elaborazione della Informazione, CNR, Pisa, Italy. In 1993 he became a postdoctoral fellow in Newcastle University, and worked on the ESPRIT projects on Predictable Dependable Computing Systems (PDCS), Design for Validation (DeVa) and on UK-funded projects on the Diversity, both in Safety Critical Software using Off-the-Shelf components. He was a member of the executive board of EU Dependable Systems of Systems (DSoS) Project, and between 2004 and 2012 headed projects on the development of a Rigorous Open Development Environment for Complex Systems (RODIN), and latterly was coordinator of the major FP7 Integrated Project on Industrial Deployment of System Engineering Methods Providing High Dependability and Productivity (DEPLOY). He now leads work on fault tolerance in Systems of Systems within the COMPASS project and is Principal Investigator of Newcastle’s Platform Grant on Trustworthy Ambient Systems.

**Suggested keywords**

SYSTEMS OF SYSTEMS
MODELLING
FAULT TOLERANCE
REQUIREMENTS ENGINEERING
TRACEABILITY
ARCHITECTURAL FRAMEWORKS
Traceable Engineering of Fault-Tolerant SoSs

Zoe Andrews¹, Jon Holt², Claire Ingram¹, Richard Payne¹, Simon Perry², and Alexander Romanovsky¹

¹ Newcastle University, UK, {firstname.lastname}@ncl.ac.uk.
² Atego, UK, {firstname.lastname}@atego.com.

Abstract. Systems of systems (SoSs) are characterised by a challenging combination of continuous evolution, emergent behaviour and distributed, autonomous, independent constituents. The development of SoSs that can tolerate faults and harmful events is hampered by these and other complexities. Currently there is little in the way of models or tools to help SoS developers to design fault-tolerant SoSs. In this paper we present a structured approach for capturing requirements for a fault-tolerant SoS and a fault modelling architectural framework (FMAF) that supports disciplined and reusable development of fault-tolerant architectures, as well as a traceable mapping of the fault-tolerant requirements into SoS architectural designs. Finally we present an example real-world SoS case study to demonstrate the application of our techniques.

1 Introduction

Systems of systems (SoSs) are characterised by continuous evolution, emergent behaviour and the presence of distributed, autonomous constituents which exhibit managerial and operational independence [1, 2]. As a result, development of SoSs that are capable of tolerating faults and other potentially harmful events (including failures of constituents) is a daunting task, hampered by the complexity of integrating heterogeneous and independent SoS constituent systems that may not have been designed for dependable integration, as well as the difficulty of communication between diverse stakeholders. The state of the art in engineering fault-tolerant SoSs lacks methods and tools to help SoS engineers engage with other stakeholders and make explicit, informed and traceable choices about the fault assumptions, the fault tolerance strategies and the redundancy to be employed.

Whilst there is a substantial body of studies on architecting fault tolerance systems (see, for example, a recent series of WADS workshops³), there are no substantial advances in developing architectural approaches to capturing faults and fault tolerance in SoSs [3] and work in this area is currently still in its infancy. Our previous work [3, 4] introduces an approach to architectural modelling of SoS fault tolerance using SysML by describing a set of views that together build a model of the erroneous behaviour and recovery procedures present in an

³ www.cs.kent.ac.uk/events/conf2009/wads/
SoS. These views fall into two categories: nominal, describing SoS structure and behaviour under the assumptions that no faults are present; and erroneous, describing possible faults, errors and failures, and the behaviour of the SoS in the presence of such dependability threats. Currently missing from the state of the art is integrated and systematic reasoning about requirements and architectures which establishes and captures traceable links between these two development steps. This paper contributes to this area by developing:

- a structured approach to capturing requirements of fault-tolerant SoSs;
- a traceable mapping of the fault tolerance requirements into SoS architectural designs; and
- an architectural framework that supports disciplined and reusable development of fault-tolerant architectures.

To ensure a wider industrial acceptance our proposed solutions are developed for the Systems Modelling Language (SysML) [5, 6] and are supported by industry-strength tools (we use Artisan Studio\(^4\) for developing prototypes and experiments). The proposed engineering techniques will allow SoS engineers to reuse their modelling experience and to support the best practice in developing dependable SoSs.

2 Background

We provide background details on requirements engineering for SoSs in Section 2.1. Section 2.2 discusses tracability and in Section 2.3 we detail an approach to defining architectural frameworks.

2.1 Requirement Engineering for SoS

Requirements engineering is a vital part of the SoS development lifecycle. The use of model-based requirements engineering (MBRE) techniques is already well-established at the systems-level [7] and therefore we hypothesise that using MBRE approaches for SoS may provide the associated benefits for SoS engineering.

The COMPASS project\(^5\) has previously proposed an approach to MBRE for SoS, extending the Approach to Context-based Requirements Engineering (ACRE) [7] to consider the characteristics typical of an SoS\(^6\). This new approach is called SoS-ACRE [9] and consists of several processes and views relating to the requirements for the engineering and management of an SoS, as shown in Figure 1.

\(^4\) www.atego.com/products/artisan-studio/
\(^5\) www.compass-research.eu/
\(^6\) The characteristics an SoS may exhibit are discussed in more detail in [8]
The SoS-ACRE processes may be summarised as follows. The SoSRequirementsEngineering Process has four processes: the SoS Requirements Development process that performs the majority of the SoS-level requirements engineering; the Requirements Elicitation Process where the initial requirements are elicited from the relevant parts of the source elements; the Context Process that defines a context at either the constituent system or SoS level; and the Verification and Validation Definition Process that defines the verification and validation criteria for the SoS.

For the management of SoS requirements, the SoS Requirements Management Process has five processes defined: the Requirements Change Process that controls changes to the constituent system or SoS requirements; the CS Process Analysis that allows the management processes of a constituent system to be understood; Requirement Control Process that ensures that all requirement changes are agreed to and commitment is obtained; the Requirements Monitor Process that allows changes in requirements at both the constituent system and SoS level to be identified; and the Traceability Process that allows traceability views to be set up.

Using processes defined for the SoS Requirements Engineering Process, requirements are identified from source elements, described in detail, contextualised in terms of the constituent systems of the SoS and its stakeholders, and validated against scenarios. Traceability views are used to trace requirements to their sources, and to the use cases described in the contexts. The processes and views are described using SysML but they are intended to be notation-agnostic.

2.2 Traceability

Traceability is concerned with capturing the relationships between artefacts created during the development process. Historically traceability has focussed on ‘requirements traceability’ (the ‘ability to describe and follow the life of a requirement, in both a forwards and backwards direction’ [10]). Research has identified the importance of pre- and post-requirements traceability (e.g., see [10, 11]) and
more recent work has extended traceability to links not involving requirements (e.g., pointed out by [12]). Traceability can be important in fault-tolerant systems for supporting validation and verification activities [13]: accurate traces enable the identification of artefacts linked to specific requirements, or of rationale behind project artefacts, to support activities such as model consistency checking [14] or test case coverage validation [15]. Our approach is intended to allow modellers to demonstrate how potential faults which have been identified at an early stage of development are handled.

### 2.3 Defining Architectural Frameworks

We present in this paper a new Architectural Framework (AF) to address the need for capturing fault tolerance aspects of SoS engineering (see Section 3). We use the COMPASS AF Framework (CAFF) to develop this new AF. The CAFF is a small architectural framework which is itself intended for use in the definition of other AFs. We provide only a brief description of the CAFF here; for full details see [16].

The CAFF is designed to address the requirements shown in Figure 2. For a full discussion of these requirements, see [16]. In summary, the CAFF is designed to ensure that:

- the needs of the AF are understood;
Fig. 3. Ontology Definition View for Architectural Frameworks

- the concepts and terms that the AF can model are defined;
- the views are identified;
- the needs for each view are understood and are related to the overall needs of the AF;
- the definition of each view is specified and based on the concepts and terms identified and
- any rules that constrain the AF are captured.

The CAFF is based on a number of concepts and relationships as described in an ontology. The ontology (Figure 3) provides a visualisation of all the key concepts pertinent to architectures and Architectural Frameworks, the terminology used to describe them and the inter-relationships between said concepts. It is based on concepts taken from [17] and [18]. The views of the CAFF are defined according to this ontology.

Two concepts illustrated in Figure 3 that are important to understand are those of Viewpoint and View. An Architectural Framework is made up of a number of Viewpoints that define the information that can be presented. When an actual architecture is developed that is based on the Architectural Framework, then one produces Views that conform to the definitions in the corresponding Viewpoint. Therefore a View is a realisation of a Viewpoint.

There are six viewpoints in the CAFF, grouped into a single Perspective, as shown in Figure 4. They are:

- The AF Context Viewpoint (AFCV), which defines the context for the AF. That is, it represents the AF concerns (requirements) in context.
The Ontology Definition Viewpoint (ODV) defines the ontology for the AF. It is derived from the AFCV.

The Viewpoint Relationships Viewpoint (VRV), which shows the relationships between the viewpoints that make up the AF and groups them into perspectives. It is derived from the ODV.

The Viewpoint Context Viewpoint (VCV), which defines the context for a particular viewpoint. That is, it represents the viewpoint concerns (requirements) in context for a particular viewpoint. It is derived from the AFCV.

The Viewpoint Definition Viewpoint (VDV), which defines a particular viewpoint, showing the viewpoint elements (and hence the ontology elements) that appear on the viewpoint.

The Rules Definition Viewpoint (RDV), which defines the various rules that constrain the AF.

The viewpoints are collected into a single perspective, the AF & Architectures Perspective, as shown by the enclosing package.

A process for using the CAFF is described in detail in [16]. In summary, the CAFF is used in the following way:

1. Define the context for the AF to be produced; understand the requirements that the AF is to address and model these by considering their use cases, using an AFCV.
2. Define the ontology that is applicable to the AF to be produced; understand the concepts and the relationships between them for the domain that the AF is to be used in and model these using an ODV.
3. Using the AFCV and the ODV, identify candidate viewpoints that cover the entire ODV. Group related viewpoints into perspectives. Capture the viewpoints, perspectives and relationships between them on a VRV.
4. For each viewpoint on the VRV, define the context for that viewpoint; understand the requirements that the viewpoint is to address and model these using a VCV.

5. For each viewpoint on the VRV, define the viewpoint in terms of the concepts and relationships from the ODV; model the viewpoint using a VDV.

6. Define any rules that apply to the AF; model such rules on a RDV.

The concepts behind considering requirements in a particular context are described in [9] and [7].

3 Fault Modelling Architectural Framework

The Fault Modelling Architectural Framework (FMAF) aims to address the need (discussed in Section 1) for a systematic approach to capturing fault tolerance and dependability aspects of SoSs. The FMAF is an extension to the informal description of fault modelling viewpoints for describing faults and recovery processes given previously [3]. The FMAF is created by applying the CAFF (Section 2.3) to these fault modelling viewpoints to define a semi-formal model and to describe the motivation behind them (see Appendix A for full model).

The FMAF supports SoS engineering in the presence of faults. Within this vast research area, it particularly focuses on providing support for:

- definition of faults, errors and failures;
- identification of fault propagation chains;
- identification of constituent systems (and the connections and interfaces between them) needed to tolerate faults;
- identification of erroneous behaviour/recovery scenarios;
- definition of erroneous behaviour and recovery processes and

- description of process behaviour in the presence of faults and of recovery process behaviour.

These are all included in the AFCV given in Figure 5.

Note that the AFCV requires that the FMAF be consistent with established dependability concepts. To satisfy this we based the ODV (see Figure 6) for the FMAF on concepts identified by Avizienis et al. [19]: a fault may lead to an error through fault activation, which in turn may lead to a failure. A failure may manifest itself as a fault as its effects are noticed higher up the system hierarchy, e.g., a failure of a constituent system is considered to be a fault of the SoS. We also use established process modelling techniques [20] as the basis of our SoS behavioural modelling: we break down the behaviour of the SoS into the behaviour within processes (low level behaviour) and the interaction between processes (high level behaviour).

The viewpoints formalised in the FMAF are those described informally in our previous work [3], summarised in Table 1. These include structural viewpoints to define the faults, errors, failures and their propagation chains as well as behavioural viewpoints that identify the behaviour of the SoS in the presence
of identified faults. As an example, consider the Fault Propagation Viewpoint (FPV), which supports the identification of fault propagation chains (see the VCV in Figure 7). This includes the definition of possible causal chains between faults, errors and failures as well as identifying the location of faults, the constituents that can detect the errors and the interfaces of the SoS at which failures occur.
Table 1. Informal description of the FMAF viewpoints

<table>
<thead>
<tr>
<th>Structural Viewpoints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>Fault/Error/Failure Definition</td>
<td>Define faults, errors and failures of the SoS. Faults, errors or failures may be generalised into abstract categories.</td>
</tr>
<tr>
<td>Fault Propagation</td>
<td>Identifies propagation of faults through errors to failures, and records relationships between the dependability threats and the constituents.</td>
</tr>
<tr>
<td>Fault Tolerance Structure</td>
<td>Shows the composition of the SoS with the required redundancy to tolerate a given fault.</td>
</tr>
<tr>
<td>Fault Tolerance Connections</td>
<td>Shows connections and interfaces between constituents of the SoS with the required redundancy to tolerate a given fault. Includes all the constituents identified in the respective Fault Tolerance Structure View.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Behavioural Viewpoints</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>Erroneous/Recovery Processes</td>
<td>Identifies the processes of the SoS, including erroneous behaviour and any required recovery processes.</td>
</tr>
<tr>
<td>Erroneous/Recovery Scenarios</td>
<td>Models behaviour in the presence of errors (with and without recovery) as scenarios. Shows erroneous behaviour propagation and recovery procedure triggers.</td>
</tr>
<tr>
<td>Fault Activation</td>
<td>Defines the behaviour within an SoS process and identifies when faults may be activated, what happens after activation and where in the process the error may be detected.</td>
</tr>
<tr>
<td>Recovery</td>
<td>Defines the behaviour of the recovery procedures that are triggered once an error has been detected.</td>
</tr>
</tbody>
</table>

may be observed. All faults, errors and failures shown in a FPV must be defined, i.e. included in a Fault/Error/Failure Definition View. A VDV is given in Figure 8, showing the view elements of an FPV and their relationships. An example of an FPV is given in Section 4.3, in Figure 12.

Supporting SoS engineering in the presence of faults is a vast and complex task. The FMAF aims to be the first step for thinking about faults in SoSs and designing dependability into the SoS. It necessarily makes abstractions to manage the complexity of the task. For example, fault activation is modelled as an event without explicitly defining how the fault is activated. This means that the same modelling approach is applicable to a wide variety of fault activation scenarios, such as the manifestation of hardware failures, internal infrastructure failures or software bugs, or even the exploitation of a vulnerability in a
Fig. 7. Viewpoint Context View for the Fault Propagation Viewpoint of the FMAF

constituent by an error (or malicious system) in the environment of the SoS. Likewise the details of how errors are detected are also abstracted away by a similar usage of error detection events. These details can of course be included in a modelling approach that applies the FMAF, but the way to do so has not (yet) been prescribed by the FMAF. It is also worth noting that a single view in a model may implement more than one FMAF viewpoint. For example, recovery processes can and do fail. Therefore fault activation and error detection may be modelled within a Recovery View, which would mean that the view is also a Fault Activation View. A final observation of the FMAF is that an approach for describing different modes of an SoS and transitions between them (e.g. transition to a degraded mode if full recovery from a fault is not possible) deserves further exploration. It is likely that extra viewpoints will be needed (to show for example the possible modes and events that cause the transitions between them) for this purpose. Extensions and further viewpoints of the model are the subject of future work.

4 Tracing Requirements to Fault Tolerant Architectures

In Section 4.1 we briefly describe the case study used to illustrate the methods proposed in this paper. In Section 4.2, we use the SoS-ACRE approach introduced in Section 2 to describe the requirements for the case study. Finally, in Section 4.3, we trace model elements of the case study to the defined requirements.
4.1 Case Study

The case study is based on a unified emergency response call centre operating in the Friuli Venezia Giulia region of North Italy, supplied by the Italian company Insiel, and is described in [3]. The SoS aims to provide aid to a specified target for any legitimate phone call made to the emergency health services. Both the nominal behaviour and erroneous behaviour (described using the FMAF viewpoints formalised in this paper) of the case study are provided in [3]. In Section 4.3 we extend some of these views to include traceability links to requirements.

4.2 Requirements

We concentrate on the requirements engineering aspect of the SoS-ACRE approach. We begin requirements modelling using the SoS requirements engineering process in which the source elements are initially identified in the Source Element View (SEV). Next we identify the constituent systems and the stakeholders of the SoS, captured in Context Definition Views (CDV) where each entity is considered as a context for the requirements elicited later. These views (and the complete set of requirements engineering views) can be found in Appendix B.

The identified constituent systems are the Phone System, Call Centre, Radio System and Emergency Response Units (ERUS). The stakeholders include the Caller, Target, Health Authority and Radio Repairman. The stakeholder diagram should be updated through the requirements engineering process, as further requirements are identified and any changes made to the SoS structure.

The next stage of the process is to signal the start of the requirements elicitation process, where initial requirements are elicited from the relevant parts of the source elements. This process results in the definition of a Requirements Definition View (RDV), as shown in Figure 9. The RDV shows the top-level
requirement of the SoS to *Provide Emergency Response* which is broken into sub-requirements which must be fulfilled. The identified requirements are traced back to the source elements (though this could conceivably form a separate diagram) to ensure there is adequate traceability of requirements. For example in the RDV in Figure 9, the *Tolerate Faults in SoS* requirement traces to the *COMPASS Description of Work* and the sub-requirements trace to *Private Communications with Insiel* and *ISADS 2013 Paper*. For this case study, this is the end of the elicitation process. In a larger study, there should be a process of identifying needs and requirements, involving multiple source elements.

On completion of the requirements elicitation process, the requirements for each identified context are considered firstly for the SoS and subsequently for each individual constituent system. We therefore begin with defining the Requirement Context View (RCV) for the SoS, placing the requirements in context – defined as use cases. Each use case should be traceable to the requirements identified in the RDV. For each use case, we identify the stakeholders of the SoS who also have some involvement or interest. This process culminates in combining related RCVs into a Context Interaction View (CIV). The CIV for an identified fault (containing only those use cases related to that fault) in the case study is given in Figure 10. This identifies which use cases the various constituents have some involvement in. It should be noted that, in SoS-ACRE, the

![Fig. 9. Requirements Definition View (RDV) for the Insiel Case Study](image-url)
meaning of the use case may be different depending on the point of view of the constituents.

The completion of this process results in the initiation of verification and validation for the different contexts. We define scenarios for each use case for a given context. We partially demonstrate this process for the CIV by defining a Validation Interaction View (VIV). Similar to the CIV, the VIV combines several Validation Views (VV) for different contexts participating in common use cases. The VIVs are defined as a semi-formal scenario as shown in Figure 11, in which an error is detected, the recovery procedure is initiated and repairs made. Several sub-VIVs are referenced and can be seen in Appendix B.

When each use case has been validated for the SoS, we return to the contexts where additional reviews are performed. The full requirements process produces RCVs for each constituent system which are subject to review and checking.
Table 2. FMAF to requirements engineering: Traceability Identification View

<table>
<thead>
<tr>
<th>Traceable Type</th>
<th>Relationship</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault/Error/Failure/Definition View</td>
<td>Requirement</td>
<td>tracesTo</td>
</tr>
<tr>
<td>Fault</td>
<td>Requirement</td>
<td>tracesTo</td>
</tr>
<tr>
<td>Error</td>
<td>Requirement</td>
<td>tracesTo</td>
</tr>
<tr>
<td>Failure detectedBy</td>
<td>Requirement</td>
<td>tracesTo</td>
</tr>
<tr>
<td>(Fault Propagation View)</td>
<td>Use case</td>
<td>tracesTo</td>
</tr>
<tr>
<td>Validation View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Fault Activation View)</td>
<td>Use case</td>
<td>tracesTo</td>
</tr>
<tr>
<td>Validation View</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error detection interruptible region</td>
<td>Use case</td>
<td>tracesTo</td>
</tr>
<tr>
<td>(Fault Activation View)</td>
<td>Validation View</td>
<td>tracesTo</td>
</tr>
<tr>
<td>Recovery View</td>
<td>Use case</td>
<td>tracesTo</td>
</tr>
<tr>
<td>Validation View</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3 Tracing Requirements to SoS Model

We provide a traceable mapping from requirements engineering elements to FMAF elements and illustrate its application using the Insiel case study. The traceability modelling makes use of the COMPASS Traceability Pattern [21], which defines four views for describing traceability:

- Relationship Identification View (RIV) – identifies the set of traceability relationships that may be used.
- Traceability Identification View (TIV) – identifies which traceable elements may partake in which traceability relationships.
- Traceability View (TV) – shows traces between traceable elements.
- Impact View (IV) – shows traceability trees for traceable elements (produced only when conducting an impact analysis).

‘Traceable elements’ may be views or view elements.

For the case study we identify a single traceability relationship \textit{tracesTo} in the RIV. This is a general traceability relationship between a view or view element in the FMAF and a view or view element from the requirements engineering model. The TIV in Table 2 shows the elements that can be traced to and from using the \textit{tracesTo} relationship. These relationships all trace backwards from FMAF elements to requirements elements (in practical usage the traces are intended to be bi-directional). It is also possible to trace between two requirements engineering elements or two FMAF elements.

The relationships identified in the TIV were applied to the Insiel case study to create TVs (see Appendix C for complete model). As an example, consider the Fault Propagation View (see Section 3) shown in Figure 12. This FPV shows a fault identified in the Insiel case study where the Radio System suffers a total failure and cannot send or receive transmissions. The fault causes an error state in which the Radio System is unavailable for use for communication between...
the Call Centre and the ERUs. During this error state it is not possible for the Call Centre to provide details of rescues to ERUs, which may then lead to a Target (e.g., a casualty) not being attended. Such an occurrence is considered to be a failure of the SoS. Traceability relationships from the TIV were added to elements of the FPV as shown in Figure 12. For example, the detectedBy dependency between the Error and the Call Centre was traced to the Detect Radio System Failure use case identified in the CIV (Figure 10) and also to a Validation View that describes detection of the Radio System Failure. Note that Figure 12 is therefore a TV as well as an FPV.

It is worth noting that further case studies may highlight the need for additional traceability relationships. Thus the traceability relationships identified in Table 2 may not be a complete list. It may also be beneficial to use stereotypes to be more specific about the types of requirements engineering elements (requirements, use cases, etc.) the FMAF elements can trace to – this will be explored in our future work.

5 Conclusion

In this paper, we have developed a structured approach to capturing requirements of fault-tolerant SoSs, a traceable mapping of the fault tolerance requirements into SoS architectural designs and an architectural framework that supports disciplined and reusable development of fault tolerant architectures. We
demonstrate these developments with a case study based on an emergency response SoS.

The FMAF describes a range of viewpoints to encompass faults, errors and failures in an SoS architectural model. Work is ongoing to define a SysML profile to complement the FMAF introduced in Section 3. The profile specialises FMAF views and view elements to include additional information through the use of stereotypes (of underlying SysML model elements) and associated tags. The case study in Section 4 is described using the profile in [4]. This profile may be used in any SysML toolset with profiling support and aims to provide support for ensuring model consistency and translation to external analysis tools.

In Section 1, we state that, to date, there have been no substantial advances in developing architectural approaches to capturing faults and fault tolerance in SoSSs. The work presented in this paper attempts to address this by amalgamating several — often separate — model-based SoS engineering activities as part of an engineering method for fault modelling. We propose that the requirements, fault-tolerant architectural models and traceability modelling efforts described in this paper be complemented through the use of fault analysis and formal verification using CML [22, 23]. In [4] we demonstrate formal verification afforded by the use of CSP and consider CML as future work.

Fault analysis is an important part of the fault modelling engineering method. Our main ongoing and future work is to develop links from architectural models, defined in SysML, to the industry-strength HiP-HOPS fault analysis tool7. As a part of this work, we define a fault analysis framework (using CAFF), define a SysML profile and provide tool support to link HiP-HOPS to the Artisan Studio SysML tool.

A consequence of defining separate frameworks for fault modelling and for fault analysis is the requirement to ensure there is a useful link between the architectural models each require. Creating consistent modelling frameworks will enable us to ensure complete traceability throughout the engineering method as has been demonstrated on the work in this paper.

Acknowledgments

The work presented here is supported by the EU Framework 7 Integrated Project “Comprehensive Modelling for Advanced Systems of Systems” (COMPASS, Grant Agreement 287829). For more information see http://www.compass-research.eu. The authors are grateful to Insiel for their input on developing the case study and to André Didier, Alexandre Mota and other members of the COMPASS project for their comments on earlier drafts.

References


7 http://hip-hops.eu
A Fault Modelling Architectural Framework Complete Model

This appendix contains all of the views that describe the FMAF (see Section 3) as SysML diagrams.

Fig. 13. AF Context View showing the requirements for the FMAF
Fig. 14. Ontology Description View relating the key concepts of the FMAF

Fig. 15. Viewpoint Relationship View relating the viewpoints of the FMAF
Fig. 16. Rules Definition View showing the rules of the FMAF

Fig. 17. Viewpoint Context View for the Fault/Error/Failure Definition Viewpoint of the FMAF
Fig. 18. Viewpoint Definition View for the Fault/Error/Failure Definition Viewpoint of the FMAF

Fig. 19. Viewpoint Context View for the Fault Propagation Viewpoint of the FMAF
Fig. 20. Viewpoint Definition View for the Fault Propagation Viewpoint of the FMAF

Fig. 21. Viewpoint Context View for the Fault Tolerance Structure Viewpoint of the FMAF
Fig. 22. Viewpoint Definition View for the Fault Tolerance Structure Viewpoint of the FMAF

Fig. 23. Viewpoint Context View for the Fault Tolerance Connections Viewpoint of the FMAF
**Fig. 24.** Viewpoint Definition View for the Fault Tolerance Connections Viewpoint of the FMAF

**Fig. 25.** Viewpoint Context View for the Erroneous/Recovery Processes Viewpoint of the FMAF
Fig. 26. Viewpoint Definition View for the Erroneous/Recovery Processes Viewpoint of the FMAF
Fig. 27. Viewpoint Context View for the Erroneous/Recovery Scenarios Viewpoint of the FMAF

Fig. 28. Viewpoint Definition View for the Erroneous/Recovery Scenarios Viewpoint of the FMAF
Fig. 29. Viewpoint Context View for the Fault Activation Viewpoint of the FMAF

Fig. 30. Viewpoint Definition View for the Fault Activation Viewpoint of the FMAF
Fig. 31. Viewpoint Context View for the Recovery Viewpoint of the FMAF

Fig. 32. Viewpoint Definition View for the Recovery Viewpoint of the FMAF
B  Insiel Requirements Engineering Complete Model

This appendix contains all of the views that were created following the requirements engineering guidelines for the Insiel case study (see Section 4) as SysML diagrams.

Fig. 33. Source Element View for the Insiel Case Study

Fig. 34. Context Definition View for the Constituent Systems of the Insiel Case Study
Fig. 35. Context Definition View for the Stakeholders of the Insiel Case Study

Fig. 36. Requirements Definition View for the Insiel Case Study
Fig. 37. Requirements Context View for the Insiel Case Study SoS
Fig. 38. Requirements Context View for the Insiel Case Study Phone System
Fig. 39. Requirements Context View for the Insiel Case Study Call Centre
Fig. 40. Requirements Context View for the Insiel Case Study Radio System

Fig. 41. Requirements Context View for the Insiel Case Study ERUs
Fig. 42. Requirements Context View for the Insiel Case Study Mobile Phone System

Fig. 43. Context Interaction View for the Insiel Case Study Radio System Failure
Fig. 44. Validation Interaction View for the Insiel Case Study Phone Voice Communications Use Case

Fig. 45. Validation Interaction View for the Insiel Case Study Answer Call Use Case

Fig. 46. Validation Interaction View for the Insiel Case Study Question Caller Use Case
Fig. 47. Validation View 0 for the Insiel Case Study Tolerate Faults Use Case

Fig. 48. Validation View 1 for the Insiel Case Study Tolerate Faults Use Case

Fig. 49. Validation Interaction View for the Insiel Case Study Detect Radio System Failure Use Case
**Fig. 50.** Validation Interaction View for the Insiel Case Study Recover From Radio System Failure Use Case

**Fig. 51.** Validation Interaction View for the Insiel Case Study Repair Radio System Failure Use Case

**Fig. 52.** Validation Interaction View for the Insiel Case Study Tolerate Radio System Failure Use Case
C Insiel Requirements Traceability Complete Model

This appendix contains all of the views that were created following the traceability pattern for the Insiel case study (see Section 4) as tables/SysML diagrams.

Table 3. Relationship Identification View for tracing from FMAF elements to requirements engineering elements

<table>
<thead>
<tr>
<th>Relationship Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tracesTo</td>
<td>Indicates a relationship between an FMAF view or view element and a requirements engineering view or view element.</td>
</tr>
</tbody>
</table>

Table 4. Traceability Identification View for tracing from FMAF elements to requirements engineering elements

<table>
<thead>
<tr>
<th>Traceable Type</th>
<th>Relationship Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault/Error/Failure/Definition View</td>
<td>Requirement</td>
</tr>
<tr>
<td>Fault</td>
<td>Requirement</td>
</tr>
<tr>
<td>Error</td>
<td>Requirement</td>
</tr>
<tr>
<td>Failure</td>
<td>Requirement</td>
</tr>
<tr>
<td>detectedBy dependency (Fault Propagation View)</td>
<td>Use case</td>
</tr>
<tr>
<td>Erroneous/Recovery Scenarios View</td>
<td>Validation View</td>
</tr>
<tr>
<td>Error detection interruptible region (Fault Activation View)</td>
<td>Use case</td>
</tr>
<tr>
<td></td>
<td>Validation View</td>
</tr>
<tr>
<td>Recovery View</td>
<td>Use case</td>
</tr>
<tr>
<td></td>
<td>Validation View</td>
</tr>
</tbody>
</table>

39
Fig. 53. Traceability View for the Fault/Error/Failure Definition View of the Insiel Case Study

Fig. 54. Traceability View for the Fault Propagation View [Radio System Failure] of the Insiel Case Study
Fig. 55. Traceability View for the Erroneous/Recovery Scenarios View [Radio System Failure] of the Insiel Case Study

Fig. 56. Traceability View for the Fault Activation View [Radio System Failure] of the Insiel Case Study
Fig. 57. Traceability View for the Recovery View [Radio System Failure] of the Insiel Case Study