COMPUTING SCIENCE

Fault Modelling for Systems of Systems

Zoe Andrews, John Fitzgerald, Richard Payne and Alexander Romanovsky

TECHNICAL REPORT SERIES

No. CS-TR-1346 August 2012

TECHNICAL REPORT SERIES

No. CS-TR-1346 August, 2012

Fault Modelling for Systems of Systems

Z. Andrews, J. Fitzgerald, R. Payne, A. Romanovsky

Abstract

This paper proposes a systematic model-based approach to the architectural description of faults and fault tolerance mechanisms in systems of systems (SoSs). The challenges of engineering dependable SoSs motivate a proposal for the view elements that would be needed to support a fault tolerance profile for SoSs using the Systems Modelling Language (SysML). The effectiveness of the approach is evaluated on a case study based on a real emergency response SoS. Results suggest that this is a promising approach, and that a comprehensive solution to the engineering of dependable SoSs requires that such a profile is linked to methods and tools for requirements elicitation, safety analysis, architectural design and formal verification.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

ANDREWS, Z., FITZGERALD, J., PAYNE, R., ROMANOVSKY, A.

Fault Modelling for Systems of Systems
[By] Z. Andrews, J. Fitzgerald, R. Payne, A. Romanovsky

Newcastle upon Tyne: Newcastle University: Computing Science, 2012.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1346)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1346

Abstract

This paper proposes a systematic model-based approach to the architectural description of faults and fault tolerance mechanisms in systems of systems (SoSs). The challenges of engineering dependable SoSs motivate a proposal for the view elements that would be needed to support a fault tolerance profile for SoSs using the Systems Modelling Language (SysML). The effectiveness of the approach is evaluated on a case study based on a real emergency response SoS. Results suggest that this is a promising approach, and that a comprehensive solution to the engineering of dependable SoSs requires that such a profile is linked to methods and tools for requirements elicitation, safety analysis, architectural design and formal verification.

About the authors

Zoe Andrews is a Research Associate in the field of dependability and formal methods in the School of Computing Science at Newcastle University. Zoe spent two years working on the ReSIST network of excellence where she was responsible for work on developing metadata-based descriptions of resilience mechanisms and providing formal support for decision making over such mechanisms. 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 is now working on the COMPASS project. In particular, she is exploring ways of modelling and analysing faults in systems of systems.

John Fitzgerald is Professor of Formal Methods in Computing Science, and Director of the Centre for Software Reliability at Newcastle. He is a specialist in the engineering of resilient systems, particularly in rigorous analysis and design tools. He leads the international COMPASS project, which is developing technology for engineering complex "Systems of Systems" and heads Newcastle's research into co-modelling and co-simulation in the design of fault-tolerant cyber-physical systems in several EU and UK-funded projects. He studied formal proof (PhD, Manchester Univ.), before joining Newcastle, where he worked with British Aerospace on the design of avionic systems in the 1990s. He went on to study the industrial application of formal modelling (specifically the Vienna Development method – VDM) as a SERC Fellow and later as a Lecturer at Newcastle. He returned to the University in 2003, having established the design and validation team in a successful SME in the embedded processor market. John is Chairman of FME, the main European body bringing together researchers and practitioners in formal methods of systems development. He is a Fellow of the BCS, and a member of the EPSRC College, the ACM and IEEE.

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.

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
DEPENDABILITY
FAULT TOLERANCE
ERROR RECOVERY
SYSML
ARCHITECTURAL DESIGN

Fault Modelling for Systems of Systems

Zoe Andrews, John Fitzgerald, Richard Payne,
Alexander Romanovsky
School of Computing Science, Newcastle University, UK
{Zoe.Andrews,John.Fitzgerald,Richard.Payne,
Alexander.Romanovsky}@ncl.ac.uk

Abstract

This paper proposes a systematic model-based approach to the architectural description of faults and fault tolerance mechanisms in systems of systems (SoSs). The challenges of engineering dependable SoSs motivate a proposal for the view elements that would be needed to support a fault tolerance profile for SoSs using the Systems Modelling Language (SysML). The effectiveness of the approach is evaluated on a case study based on a real emergency response SoS. Results suggest that this is a promising approach, and that a comprehensive solution to the engineering of dependable SoSs requires that such a profile is linked to methods and tools for requirements elicitation, safety analysis, architectural design and formal verification.

Keywords: systems of systems, modelling, dependability, fault tolerance, error recovery, SysML, architectural design.

1 Introduction

Developing dependable computing systems is notoriously difficult, in part because of the complexity introduced by the need to detect and recover from errors. As a result, fault tolerance is often the least well specified, verified, debugged, tested and documented aspect of a system's design. Consequently, the introduction of error detection and system recovery can even reduce dependability.

Engineering dependable systems of systems (SoSs) is even more daunting. SoSs are systems built, at least in part, from pre-existing constituent systems that may not have been designed for integration. Their characteristics include autonomy, dynamic connectivity, distribution, operational and managerial independence of constituents, evolution over time, and emergent behaviour [10, 3]. Examples include transport (e.g. integrating multiple air traffic control and airline systems to deliver safe and reliable services), and smart power grids (e.g. integrating multiple generation, storage and delivery systems). The independence of the constituent systems of an SoS means that novel fault tolerance techniques are required compared to situations in which all the components can be readily commanded.

As reliance is placed on SoSs, it becomes vital to have methods of verifying their dependability properties. For this, it is paramount to have a precise model of the SoS that supports the trade-off of alternative designs at early development

stages and the determination of the contracts between constituent systems. In this way, requirements on each constituents provider are clear, and existing systems can be adapted or wrapped for participation in the SoS. Model-based methods are increasingly used as a means to manage and control the overall complexity of a system, reveal and document its key structure and behaviour, and communicate these to stakeholders. Models for the realization of enterprise, system, and software architectures can be visualised by inter-connected views that allow different aspects of the underlying systems to be analysed.

In this paper we focus on model-based methods for the systematic architecting of fault tolerance measures in dependable SoSs. The main contribution is to propose an approach based on a set of views for modelling fault tolerance using the Systems Modelling Language (SysML [18]). The need for such an approach is motivated by the challenges of engineering dependable SoSs (Section 2). Starting from a baseline of work on dependability and disciplined modelling in SysML (Section 3), we propose architectural views to model faults and fault tolerance mechanisms for SoS (Section 4). A case study based on a real emergency response SoS (Section 5) provides an initial evaluation of the views (Section 6) and leads to a discussion of how they could be developed towards a full SysML profile for fault tolerant SoS. Conclusions and future work are described in Section 7.

2 Background and Related Work

In Section 1, we argued that the engineering of dependable SoS necessitates a precise, model-based approach. In this section, we introduce basic concepts and terms used in our work, which relates systems of systems (Section 2.1) to dependability, fault tolerance (Section 2.2) and architectural modelling (Section 2.3) and argue that such an approach requires models of SoS architecture as well as functionality.

2.1 Systems of Systems

The term *systems of systems* refers to systems composed of existing *constituent systems* in such a way that the SoS provides new emergent functionality [10, 3]. Management of the complex interactions between constituent systems may achieve savings across a whole collaborative activity or business, but in order to take advantage of the opportunities afforded by SoS technology, it is necessary to address the challenges that stem from the managerial and operational independence of the constituents, their network and geographical distribution, the reliance placed on emergent behaviour, their heterogeneity, and their capacity to evolve during the SoS's life.

SoS engineering is now a lively area of research. Our own work is part of an EU initiative supporting a suite of research, community building and roadmapping projects[1]. There are numerous reports on the application of SoS engineering in areas including defence, transport, power grid, health care, space

[1]Our work focuses on dependability and formal model-based design methods for SoS in the COMPASS project (`www.compass-research.eu`). Related projects include DANSE (`www.danse-ip.eu`), T-AREA-SOS which promotes transatlantic cooperation (`www.tareasos.eu/` and ROAD2SOS (`www.road2sos-project.eu`).

and robotics. However, methods for engineering dependability into SoS from the architectural design stage onwards remain in their infancy.

2.2 Dependability and Fault Tolerance

The *dependability* of a system is defined as its ability to deliver a *service* that can be justifiably trusted [1]. The means for attaining dependability can be broadly classified into fault *prevention*, *tolerance*, *removal* and *forecasting*. Ensuring dependability usually requires various techniques in different phases of the life cycle. Systematic approaches to structuring requirements, architectural design and system verification can all assist fault prevention and removal. However, there are numerous sources and types of faults, many of which cannot be completely eliminated in development. For example, it may be infeasible to produce defect-free software, hardware may degrade, and users or the system environment may behave in unanticipated ways.

We use the dependability taxonomy established in [1] which defines a *system failure* as a deviation of the service provided by the system from expected (correct) behaviour. An *error* is defined as the part of the system state that can lead to its subsequent service failure. The adjudged or hypothesized cause of an error is called a *fault*. An erroneous state is created when a fault is triggered, this can lead to changes in the system service (its external behaviour) – this is classified as a system failure (inability to deliver the service). Fault tolerance measures (the focus of this paper) prevent failures from arising in the presence of faults; this is achieved by detecting *errors* and conducting system *recovery* to remove the erroneous state and, if possible, faults.

2.3 Fault Tolerance Architectures in SoSs

The characteristics of SoSs make the need for fault tolerance clear. There is a risk of unanticipated failures in independently managed constituent systems. The distributed nature of the SoS can introduce communications faults. There is dynamic evolution (change) in constituents, and the potential for mismatches between constituents. There is also a high risk of concurrent error caused by error propagation or by reliance of several constituent systems on the same components/infrastructures. Any of these can lead to failure of the SoS as a whole. This begs a crucial question: where is the boundary of an SoS? More specifically, it is not always clear to which extent we could recover or involve in recovery constituent systems when we are recovering in an SoS. Together, these sources of risk suggest that the introduction of fault tolerance into SoS designs requires a model that identifies SoS boundaries, constituents and their connectors. These aspects form the *architecture* of the SoS.

The majority of work on architecture description languages (ADLs) is at the level of software architecture [11]. Many existing notations (such as UML [12] and formal ADLs such as Darwin [9]), therefore, do not contain architectural abstractions suitable for modelling SoSs. The advancement of model-based approaches to embedded systems has strengthened the need for notations modelling both hardware and software elements. Recent notations including SysML [13] and AADL [7] address architectures at the system engineering level and may be considered for SoS descriptions [15].

There is a substantial body of work on architecting fault tolerant systems (see, for example, a recent series of WADS workshops[2]), including work on developing an Error Model Annex to AADL that allows architectural modelling of dependability features [17], focusing on the error transitions of components and how they are propagated. A number of fault tolerance patterns (such as error detection mechanisms in a hot standby redundant system) have also been defined in AADL [17]. In addition, UML has been used for modelling erroneous behaviour in embedded systems [2, 4]. Unfortunately, there are no substantial advances in developing architectural approaches to the description of faults and fault tolerance in SoSs, and most current modelling notations lack precise foundations to support the required trade-off studies through which particular SoS models can be assessed [20].

3 Baseline

Our work aims to provide precise model-based methods and tools for the description and analysis of architecture and functionality, faults, errors and failures, as well as fault tolerance mechanisms. Within the overall design flow, we aim to allow the early construction of abstract models including faults and fault tolerance, once dependability requirements begin to be captured. Such models should help developers to make informed decisions, comparing alternative designs, and verifying that the SoS has enough fault tolerance to meet the dependability requirements and adjust the architecture where necessary.

Our approach combines the disciplined use of SysML for architectural description with a formal language (the COMPASS Modelling Language – CML) for the detailed specification of functional and other characteristics of constituent systems. The independence of constituent systems means that these interfaces are described in contractual terms, recording the assumptions made by constituent systems and the guarantees that they offer when those assumptions are met. An automated translation from the architectural model in SysML into CML, coupled with the formal semantics of CML, together provide a basis for the composition and refinement of interface descriptions and enables automated analysis of SoS properties by proof, model checking and/or simulation. The work reported here focuses on modelling faults and fault tolerance mechanisms in this setting. It is intended to form part of an integrated approach combining dependability requirements capturing and traceability, fault tolerance modelling, safety analysis, architectural design of fault tolerance and formal verification of the fault tolerance and dependability properties. The elements of CML are described in greater detail elsewhere [21]. In the remainder of this paper, we focus on the architectural aspects, and hence on SysML.

Not all types of error recovery are likely to be suitable for SoSs. The most popular mechanisms based on backward error recovery (such as rollback, abort, retry, checkpointing) or replication are not readily applicable because of the problems of distribution and the independence of constituents. It is clear that forward error recovery (in the form of exception handling or compensation) is the most appropriate recovery to be conducted at the level of SoSs. The emergent behaviour of the SoS relies on the coordination of its constituents, so SoS failures are likely to require cooperative recovery involving several constituent systems

[2] `http://www.cs.kent.ac.uk/events/conf/2009/wads/`

because of the difficulty of containing errors when constituent systems heavily depend on each other.

SysML [18] is a profile for UML 2.0, designed for systems engineering, with several features giving it the potential for SoS architecture description. SysML uses a subset of UML 2.0 and provides further extensions to the UML superstructure. It has wide industrial support and a sound tool base. It provides several diagram types, with 'precise natural language' semantics, to support the description of SoS architectural structure, behaviour and requirements. Some extensions have been made to SysML for fault modelling or safety analysis. However, these tend to focus on specific goals such as the identification and refinement of (non-functional) requirements [16, 19] in safety critical systems, or the extraction of component dependencies for fault tree analysis [22]. The closest approach to ours extends SysML with stereotypes for the purposes of automated extraction of Failure Modes and Effects Analysis [6]. However, we believe that the approach we propose is the first view-based approach to fault modelling in SysML. Such an approach provides not just a notation but also a methodology for describing and understanding the impact of faults and recovery procedures in SoSs. Our focus is on demonstrating how to separate normal and abnormal behaviour of a SoS at the architectural level and how to use the SysML features to capture error detection and recovery concerns.

4 Architectural Modelling of Fault Tolerance using SysML

SysML is a rich language with many different ways to model the same behaviour. The SysML diagrams provide views onto the underlying model describing the architectural structure, behaviour and requirements of the model. Rather than prescribing a subset of diagrams that are suitable for fault modelling, we opt to describe a set of views that together will build up a picture of the erroneous behaviour and recovery procedures present in an SoS. These views fall into two categories:

Nominal The structure and behaviour of the system under the assumption that no faults are present.

Erroneous Behaviour & Recovery Describes possible faults, errors and failures, and the behaviour of the SoS in the presence of such dependability threats. Also includes the structure and behaviour of the system with recovery procedures included to deal with the identified faults and errors.

We propose the initial modelling of the nominal structure and behaviour of the SoS using the views outlined in Table 1. For each view, we provide a brief description. In Section 7, we propose future work to enhance their definition.

Several views are proposed for modelling erroneous behaviour and recovery procedures. Some of these show the structural relationships between faults, errors, failures and components, whilst others show how faults and errors impact on the behaviour of the SoS. For the sake of clarity, most of these views (with the exceptions of Fault/Error/Failure Definition and the Recovery Processes views) are intended to focus on one fault at a time, although for interacting faults it

Structural Views	
Name	**Description**
Ontology	Defines the concepts of the SoS, with their relationships provided. Includes structural and behavioural concepts, defined using block definition diagrams (BDDs).
Composition	Composition, association and generalisations of constituents. Defined using BDDs, may contain operations and attributes.
Connections	Connections between constituent systems, and between the SoS and its environment. Connections defined using internal block diagrams (IBDs) and may identify interfaces between constituents.
Behavioural Views	
Name	**Description**
Scenarios	Describes sequences of interactions between constituent systems and the SoS and its environment defined using sequence diagrams (SDs). Do not define the complete SoS behaviour.
Processes	Processes are sequences of actions to be taken in different stages of the SoS lifecycle [8]. A combination of BDDs and activity diagrams (ADs) are used in process modelling.

Table 1: SysML views of nominal structure and behaviour

may be useful to model several faults in a single view. The proposed structural and behavioural views are outlined in Table 2.

5 Case Study: an Emergency Response SoS

A case study based on a unified emergency response call centre is used to illustrate the fault modelling approach and evaluate the applicability of the views described in Section 4. The aim is to assess (Section 6) what additional understanding of the SoS may be gained by using these views and how well they can capture the behaviour of the SoS in the presence of faults.

The case study is supplied by the Italian company Insiel to operate in the Fruili Venezia Giulia region of North Italy. The system of interest provides an emergency response to targets identified by the public: we refer to this system as the Insiel SoS. The Insiel SoS may be classified as an *acknowledged* SoS the terms of Maier [10] and Dahmann et al. [5]. The constituent systems are operationally and managerially independent – provided and developed by external organisations. Each constituent may be geographically distributed and may evolve. The SoS is considered acknowledged as the SoS has recognised objectives and a recognised manager in the form of the call centre, whilst the constituents retain their independence. Evolutionary changes are based on collaboration between the SoS and its constituents.

5.1 Informal Description of Case Study

Figure 1 provides a high-level, informal structural representation of the ER system. The figure identifies the SoS boundary and the constituent systems: *Phone System*, *Call Centre*, *Radio System* and *Emergency Response Unit (ERU)*. The

Structural Views	
Name	**Description**
Fault/Error/Failure Definition	Define faults, errors and failures of the SoS using BDDs. Faults, errors or failures may be generalised into abstract categories.
Fault Propagation	Identifies propagation of faults through errors to failures and their relationships to constituent systems using IBDs.
Fault Tolerance Structure	Extends nominal *Composition View* with additional components required to tolerate a given fault.
Fault Tolerance Connections	Extends nominal *Connection View* with additional components identified in the FTS view and the interfaces and connectors to tolerate a given fault.
Behavioural Views	
Name	**Description**
Erroneous/Recovery Scenarios	Models behaviour in the presence of errors (with and without recovery) as scenarios in SDs. Shows erroneous behaviour propagation and recovery procedure triggers.
Erroneous/Recovery Processes	Extends nominal *Process* BDDs to include behaviour resulting from faults. Further processes are added to model the recovery procedures.
Fault Activation	Extends nominal *Process* ADs to model the low level behaviour of errors. Identifies when faults may be activated, what happens after activation and where in the process the error may be detected.
Recovery	Further extends nominal *Process* ADs to show the behaviour of the recovery procedures once an error has been detected.

Table 2: SysML views of erroneous/recovery structure and behaviour

entities in the environment with which the SoS interacts include *Callers* and *Targets*. The purpose of this simplified Insiel SoS is to meet one high-level requirement: for every call received, send an ERU with correct equipment to the correct target.

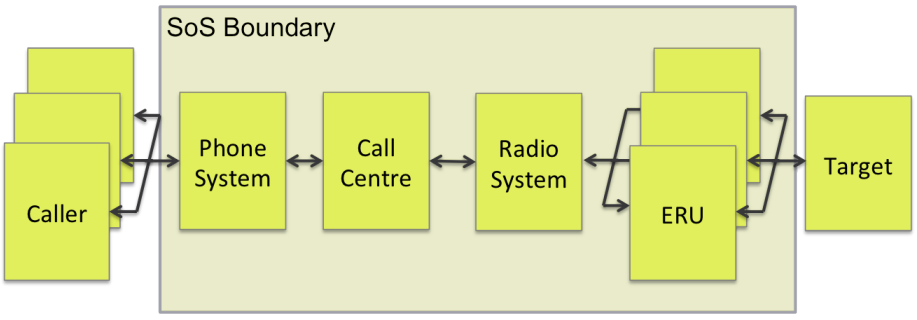

Figure 1: Outline structure of Emergency Response SoS, its constituent systems and environment

Fault	Description	Error Detected by	Recovery
Fault 1	Complete failure of the Radio System	ERU and/or Call Centre	The ERU driver uses his/her mobile phone. If there is no mobile phone coverage, the personnel uses landline phones (e.g. the phone of the patient)
Fault 2	An ERU breaks down or crashes	ERU (Driver) or Call Centre	ERU with patient: a new ERU without medical personnel is sent, to retrieve the patient and the ERU crew. ERU driver waits for the tow truck ERU without patient: a new ERU or a car is sent to retrieve ERU crew (except driver, as he waits for assistance)
Fault 3	An Operator sends an ERU to the wrong location	Call Centre (Operator), Caller or ERU	Re-direction of the ERU to the correct target location (assuming that this can be resolved from the information available), or select closer ERU

Table 3: Emergency Response SoS faults of interest

The *Phone System* is a complex system, operated and managed by several external telecommunications companies. It provides different types of emergency phone numbers corresponding to different types of emergency (e.g. fire, public disorder, mountain rescue) which may require a specialised response. Each incoming call from caller is given unique identifier and routed to the Call Centre. The *Call Centre* constitutes human operators and the call centre software system. Given an incoming call, the Call Centre generates and manages rescue events. The *Radio System* handles communications between the Call Centre and ERUs, maintaining a database of ERU identifiers to ensure correct message routing. Finally, the *ERUs*, with externally managed and owned communication systems, provide aid to a specified target. ERUs may contain specific equipment for particular type of aid.

5.1.1 Faults of Interest

For our study, we identify three faults which may occur in the simplified Insiel SoS. These relate to failures of the constituent systems leading to error states in the SoS, which are discovered by different constituents. For each fault, we propose recovery guidelines which may involve several constituent systems and reconfiguration of the SoS architecture where required. The faults are outlined in Table 3. We also explored the recovery procedures required in the presence of multiple simultaneous (concurrent) errors and found that there was no need for joint recovery (where multiple errors need to be resolved at the same time). However, it was observed that the Radio System is an essential supporting system for the SoS. This means that recovery from a Radio System failure needs to be completed before recovery from (concurrent) errors caused by the activation of Fault 2 or Fault 3 can be handled.

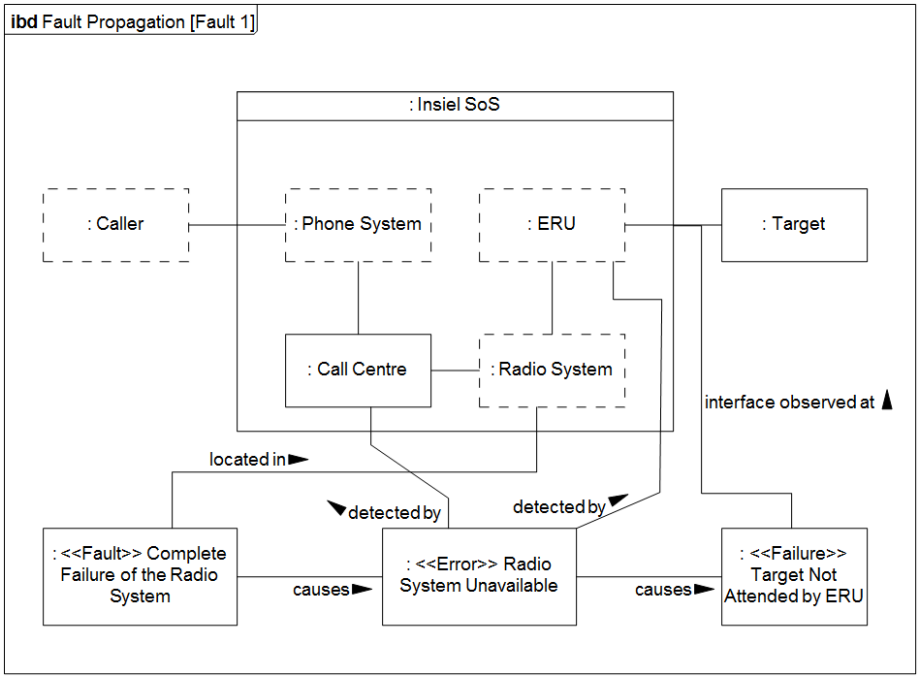

Figure 2: Fault Propagation View for Fault 1

5.2 Modelling the Case Study in SysML

The SysML views described in Section 4 were applied to the emergency response case study. The purpose of this is two-fold, first to illustrate their usage and second to evaluate their applicability to a real life SoS. Some views are omitted here, the complete set of SysML views can be found in Appendix A. Here we present the *Fault Propagation View*, the *Fault Tolerance Connections View* and the *Fault Activation View*. The fault shown in all of these views is *Fault 1*, the complete failure of the Radio System as described above.

Figure 19 shows the *Fault Propagation View* for *Fault 1*. The fault "Complete Failure of the Radio System" leads to the error state "Radio System Unavailable", which in turn leads to the SoS failure "Target Not Attended by ERU" (as the call centre cannot communicate with the ERU to tell it where to go). This causal chain is shown in the view as well as links from the faults, errors and failures to the constituent systems. For example, it can be seen that the error state may be detected either by the Call Centre or by an ERU.

In Figure 21 we start to consider how to tolerate the Radio System failure in the *Fault Tolerance Connections View* for *Fault 1*. This view shows the SoS architecture that is required to tolerate the identified fault. Constituent systems, interfaces and/or connections (that allow the SoS to handle the fault) are added to the *connections view* of the nominal behaviour. For the emergency response system an extra constituent system (the *Mobile Phone System*) has been incorporated into the SoS. This provides the same functionality as the Radio System, effectively acting as a hot spare. Note also that a new connection

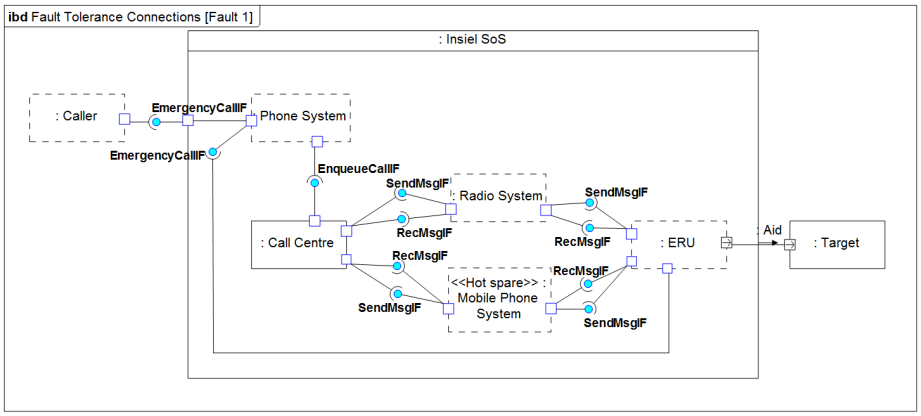

Figure 3: Fault Tolerance Connections View for Fault 1

has been added between the ERU and the Phone System. This allows the situation where an ERU communicates to the call centre using a landline phone. The interfaces between the components are very simple with just one operation of the same name. For example the *SendMsgIF* has the operation

$$sendMsg(in\ senderId : ID, in\ destnId : ID, in\ msg : String)\ .$$

Interface operations are usually shown in a separate SysML BDD for clarity.

Figure 24 illustrates the *Fault Activation View* of the SoS. The behaviour of the SoS is extended to show when a given fault may be activated (using interruptible regions) and how this impacts on the actions performed. The opportunities for detecting errors resulting from the fault activation are also shown using interruptible regions. Figure 24 shows the "initiate rescue" process of the emergency response SoS, which involves processing the data from the caller, allocating an ERU and requesting the ERU to service the rescue event. The first interruptible region encountered in the AD shows where *Fault 1* can be activated. This results in the message from the Call Centre being dropped (an omission fault), which may then be detected in the second interruptible region. If the error is detected a recovery procedure is started, otherwise the process terminates without an ERU being sent to the target (resulting in a failure of the SoS).

6 Discussion

6.1 Case Study Evaluation

The case study provided a useful SoS with which to validate the SysML modelling approach proposed in Section 4. In producing the SysML views of the emergency response SoS, several questions arose that may otherwise not have been considered until later in the development process. For example, consider the situation where ERU personnel use a landline phone to contact the call centre (because the radio and mobile systems are unavailable). Producing the fault

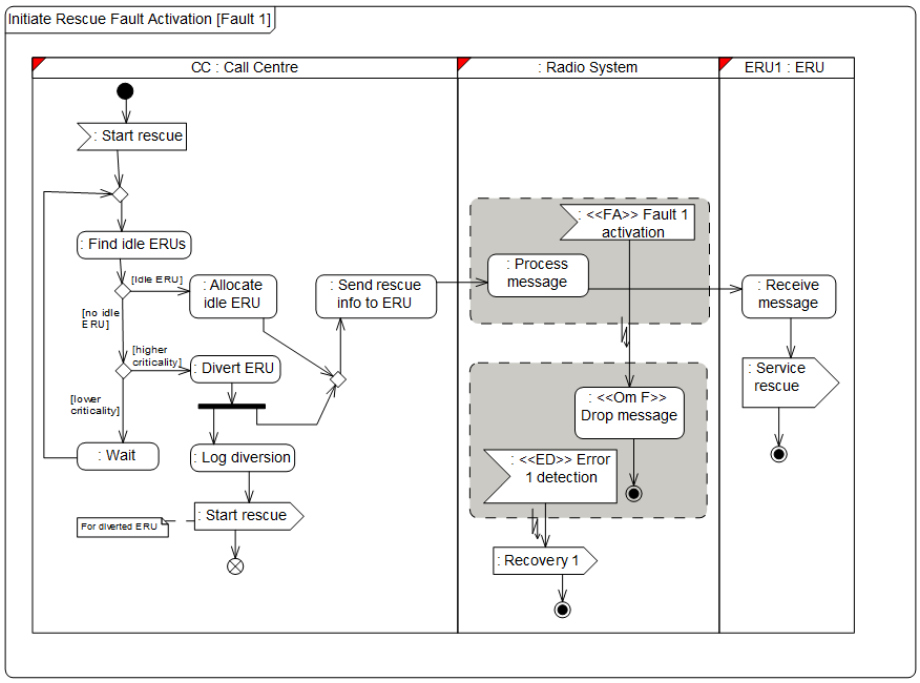

Figure 4: Fault Activation View (initiate rescue process) for Fault 1

tolerance connections view for this scenario raised the question as to whether the ERU personnel use the same interface as the general public, or whether they have a direct line to call centre staff (note that the former of these was modelled in Figure 21). The case study clearly demonstrated that in a complex SoS it is possible to recover from a failure of a constituent system using reconfiguration and involving alternative solutions provided by another constituent system.

Some of the SysML elements shown in Section 5 include text inside guillemets. These highlight features (e.g. $\langle\langle FA \rangle\rangle$ for fault activation events) of the models that may be more generally applicable for fault modelling in SysML. These special elements can be described in a SysML "profile", discussed in more detail below.

There were several challenges in modelling the case study in SysML. For example, it is unclear how best to record that an event or action should have occurred, but did not due to some fault. In ADs the interruptible regions provide a useful mechanism for indicating that something has gone wrong, but at a higher level of abstraction (SDs) it is less clear how to represent this. A possible solution to this is to "stereotype" events and actions as failure events or omission faults as discussed below. A further issue is how to make this approach scalable to the real complexity of SoSs (whilst the case study is a real SoS, it was modelled at a very high level of abstraction). Using sub-activities in ADs is one possible solution to this, but it is not immediately obvious how to represent the propagation of faults at a lower level of abstraction to the parent AD.

Finally, an important goal of the COMPASS project is to enable safety analysis such as fault tree analysis or HiP-HOPS [14]. Whilst we feel that the

views described above aid in the understanding of the erroneous behaviour of a SoS, it is not yet clear how the models can be exploited to provide input to some suitable safety analysis method.

6.2 Towards a SysML Profile

The SysML views for fault modelling outlined in Section 4 were described informally. In future work it is our intention to define these more formally using a SysML *profile*, a mechanism for extending SysML for different purposes primarily through the use of stereotypes [18]. Stereotypes allow the specialisation of diagrams (e.g. to define views) and diagram elements (e.g. to define special elements such as those highlighted by guillemets in the emergency response case study) by allowing additional properties of these elements to be defined.

A number of diagram elements have been identified as suitable candidates for stereotypes in a fault modelling profile (based on the views described in Section 4). Faults, errors and failures (in the structural views) could be represented by stereotyped blocks with properties such as "located in" (fault), "detected by" (error) and "observed at" (failure). The fault/error/failure propagation chain could be included in these stereotypes using "caused by" and/or "causes" properties. Processes and recovery processes could be stereotyped for the *Recovery Processes View*. This would simply highlight which processes are normal behaviour and which provide fault tolerance. Actions within the processes (and ADs) may be tagged as erroneous (should not occur in normal operation) or omission (highlighting where something should have occurred but did not due to a fault). Events in SDs and ADs could be stereotyped as well for fault activation, error detection and failure occurrences. To complete the fault modelling profile in SysML, each view would be defined as a stereotype of its base diagram (e.g. BDD for the *Fault/Error/Failure Definition View*) and the diagram elements that could appear in the view (including the new stereotyped elements) would be specified.

7 Conclusions

The approach presented in this paper supports disciplined SysML modelling of fault tolerant systems of systems. It is based on a clear separation of SoS normal and erroneous/recovery behaviour, and supports explicit reasoning about SoS faults and errors, error propagation, fault and error handling. This work contributes to our overall aim of developing a general approach to systematic development, modelling and verification of dependable SoSs. To fully achieve this aim, we need to rigorously define a fault tolerance SysML profile for modelling of SoSs. This will enable a semi-automatic translation from fault tolerance SysML models to CML models to allow formal verification of the dependability (in particular, fault tolerance) properties. A method for requirements elicitation (supported by SysML) that precedes the architectural design (as presented in this paper) is also required. Finally, a link between the SysML architectural design and established safety analysis methods needs to be developed for safety critical SoSs. These directions form the next steps of our research in this area.

Is our approach specific to systems of systems, or is it just as applicable to systems generally? SoSs form a subclass of systems with particularly challenging

properties, so it is not surprising if the set of views that we propose could be deployed more widely. We would, however, expect that the more limited range of recovery mechanisms applicable in the SoS case would affect the way in which the set of views might be used in the SoS setting.

Acknowledgements

The authors' work is supported by the EU FP7 Project COMPASS (No.287829), and the EPSRC Platform Grant on Trustworthy Ambient Systems. The authors are grateful to Enrico Fracasso of Insiel for his input in developing the case study and to Alexandre Mota, André Didier, Jon Holt and Simon Perry for their feedback on earlier versions of the paper.

References

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1:11–33, 2004.

[2] Simona Bernardi and José Merseguer. A UML profile for dependability analysis of real-time embedded systems. In *Proceedings of the 6th international workshop on Software and performance*, WOSP '07, pages 115–124, New York, NY, USA, 2007. ACM.

[3] John Boardman and Brian Sauser. System of Systems – the meaning of "of". In *Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering*, Los Angeles, CA, April 2006. IEEE.

[4] Andrea Bondavalli, Mario Dal Cin, Diego Latella, István Majzik, András Pataricza, and Giancarlo Savoia. Dependability analysis in the early phases of UML-based system design. *Comput. Syst. Sci. Eng.*, 16(5):265–275, 2001.

[5] Judith S. Dahmann, George Rebovich Jr., and Jo Ann Lane. Systems engineering for capabilities. *CrossTalk Journal (The Journal of Defense Software Engineering)*, 21(11):4–9, November 2008.

[6] Pierre David, Vincent Idasiak, and Frédéric Kratz. Reliability study of complex physical systems using SysML. *Reliability Engineering System Safety*, 95(4):431–450, 2010.

[7] Peter Feiler, David Gluch, and John Hudak. The architecture analysis and design language (AADL): An introduction. Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, 2006.

[8] Jon Holt. *A Pragmatic Guide to Business Process Modelling*. British Computer Society, Swinton, UK, UK, 2nd edition, 2009.

[9] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. Specifying distributed software architectures. In *Proceedings of the 5th European Software Engineering Conference*, pages 137–153. Springer-Verlag, 1995.

[10] Mark W. Maier. Architecting Principles for Systems-of-Systems. *Systems Engineering*, 1(4):267–284, 1998.

[11] Nenad Medvidovic, Eric M. Dashofy, and Richard N. Taylor. Moving architectural description from under the technology lamppost. *Information and Software Technology*, 49(1):12–31, Janruary 2007.

[12] Object Management Group OMG. Unified Modelling Language infrastructure version 2.2. http://www.omg.org/spec/UML/2.2/Infrastructure, February 2009.

[13] Object Management Group OMG. Systems Modelling Language version 1.2. http://www.omg.org/spec/SysML/1.2, June 2010.

[14] Y. Papadopoulos, M. Walker, D. Parker, E. Rüde, R. Hamann, A. Uhlig, U. Grätz, and R. Lien. Engineering Failure Analysis and Design Optimisation with HiP-HOPS. *Engineering Failure Analysis*, 18:590–608, 2011.

[15] Richard J. Payne and John S. Fitzgerald. Evaluation of Architectural Frameworks Supporting Contract-based Specification. Technical Report CS-TR-1233, School of Computing Science, Newcastle University, December 2010.

[16] J.-F. Pétin, D. Evrot, G. Morel, and P. Lamy. Combining SysML and formal models for safety requirements verification. In *ICSSEA 2010 — 22nd International Conference on Software & Systems Engineering and their Applications*, 2010.

[17] Ana-Elena Rugina, Peter H. Feiler, Karama Kanoun, and Mohamed Kaâniche. Software dependability modeling using an industry-standard architecture description language. *CoRR*, abs/0809.4109, 2008.

[18] Systems Modeling Language (SysML) Specification. Technical Report Version 1.2, SysML Modelling team, June 2010. http://www.sysml.org/docs/specs/OMGSysML-v1.2-10-06-02.pdf.

[19] Kleanthis Thramboulidis and Sven Scholz. Integrating the 3+1 SysML view model with safety engineering. In *Proceedings of 15th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2010, September 13-16, 2010, Bilbao, Spain*, pages 1–8. IEEE, 2010.

[20] Ricardo Valerdi, Elliot Axelband, Thomas Baehren, Barry Boehm, Dave Dorenbos, Scott Jackson, Azad Madni, Gerald Nadler, Paul Robitaille, and Stan Settles. A Research Agenda for Systems of Systems Architecting. *Int. J. System of Systems Engineering*, 1(1/2):171–188, 2008.

[21] Jim Woodcock and Alvaro Miyazawa. Features of CML: a Formal Modelling Language for Systems of Systems. In *Proceedings of the 7th IEEE Conference on System of Systems Engineering (SOSE 2012)*, Genoa, Italy, July 2012. IEEE.

[22] Jianwen Xiang, Kazuo Yanoo, Yoshiharu Maeno, and Kumiko Tadano. Automatic synthesis of static fault trees from system models. In *Fifth International Conference on Secure Software Integration and Reliability Improvement*, pages 127–136. IEEE Computer Society, 2011.

A Case Study Models

This appendix provides the SysML views created for the emergency response case study described in Section 5.

A.1 Nominal behaviour

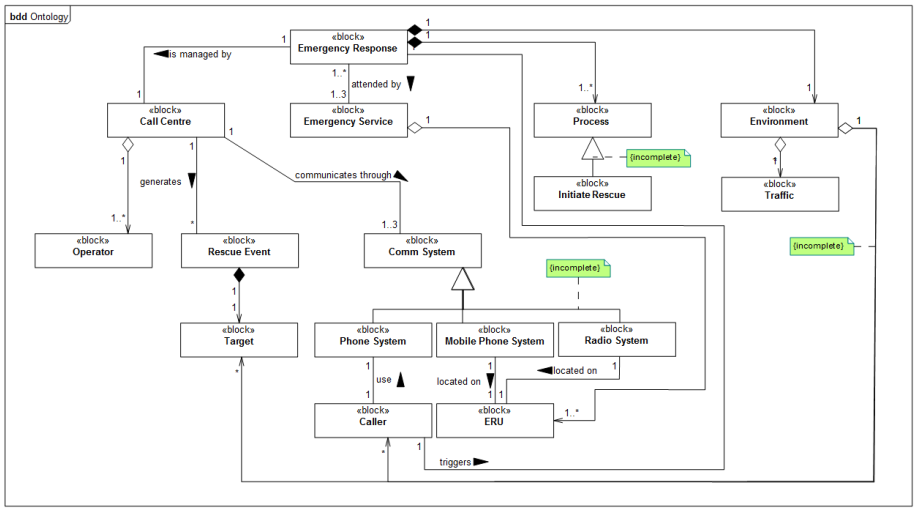

Figure 5: Ontology View

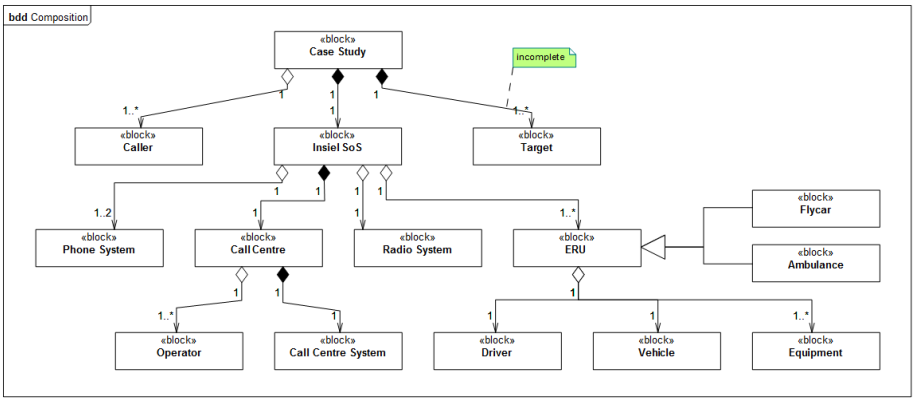

Figure 6: Composition View

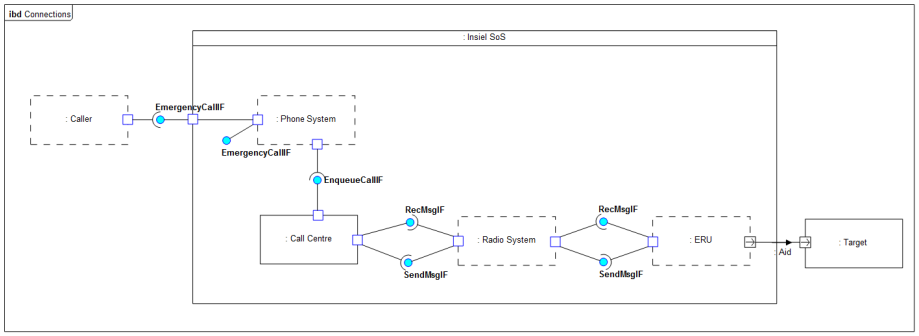

Figure 7: Connections View

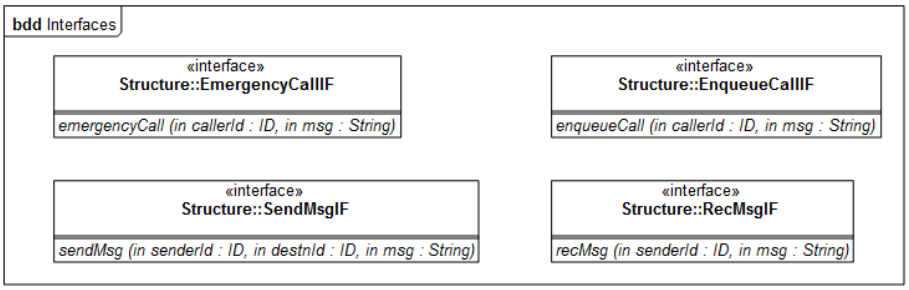

Figure 8: Connections View (interfaces)

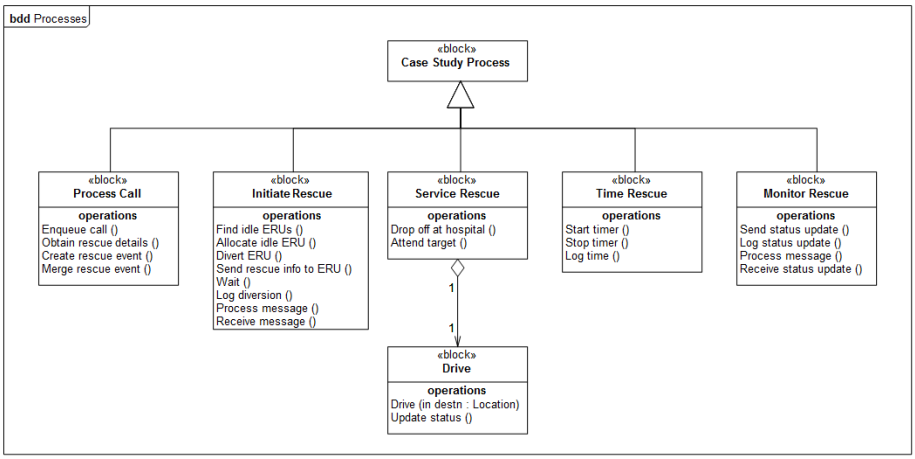

Figure 9: Processes View (BDD)

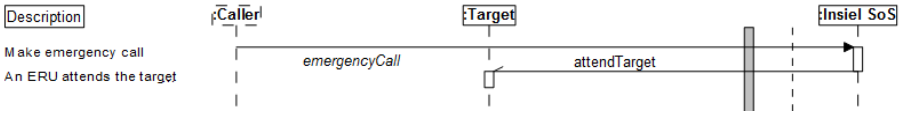

Figure 10: Scenarios View (SoS level)

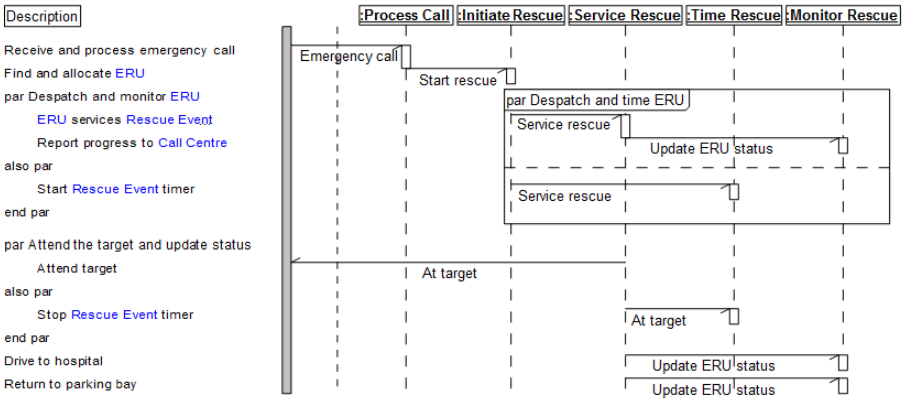

Figure 11: Scenarios View (internal level)

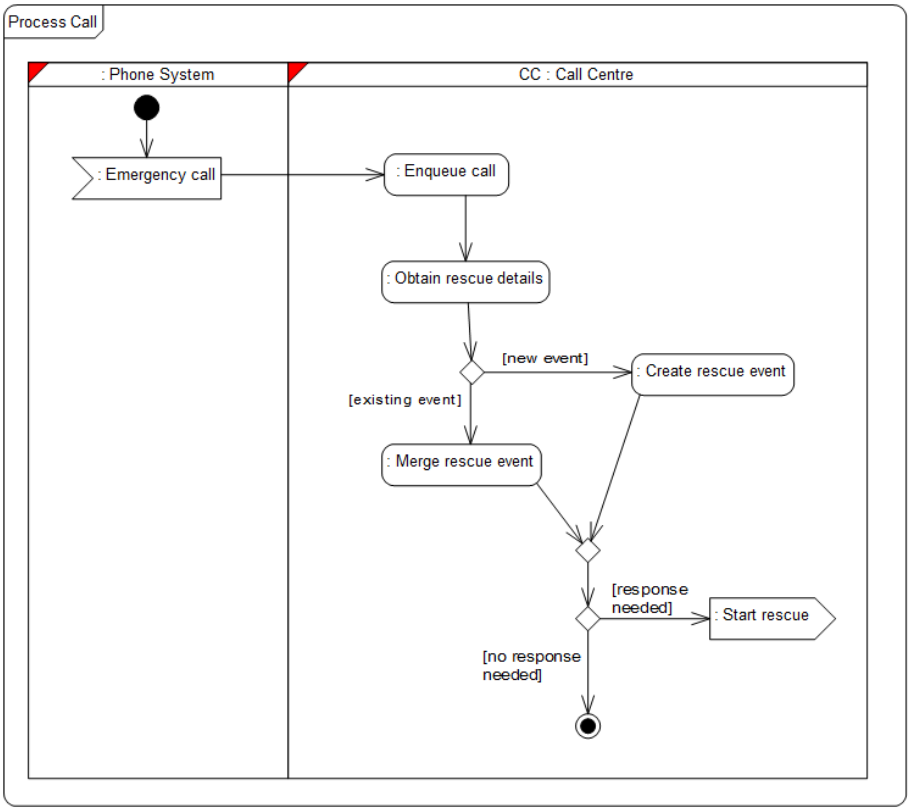

Figure 12: Processes View (process call process)

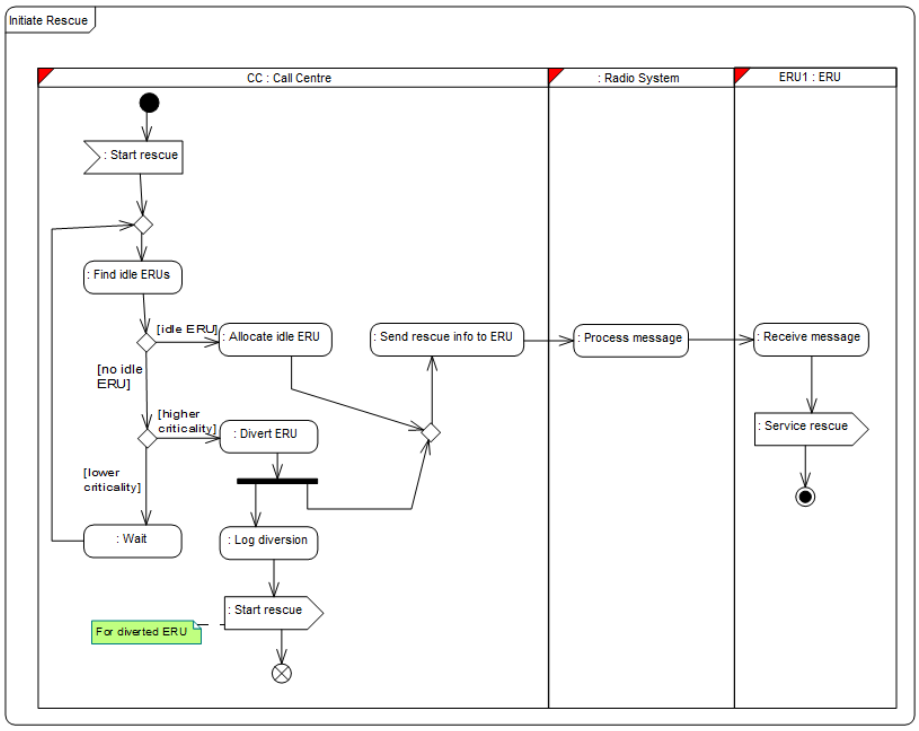

Figure 13: Processes View (initiate rescue process)

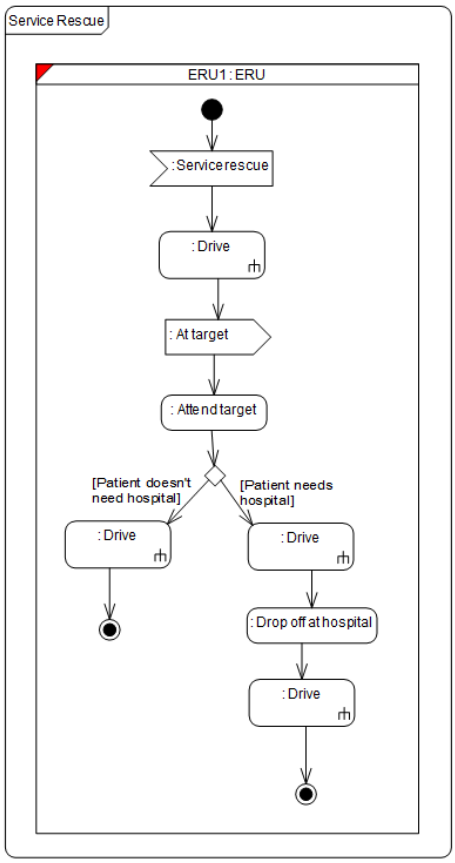

Figure 14: Processes View (service rescue process)

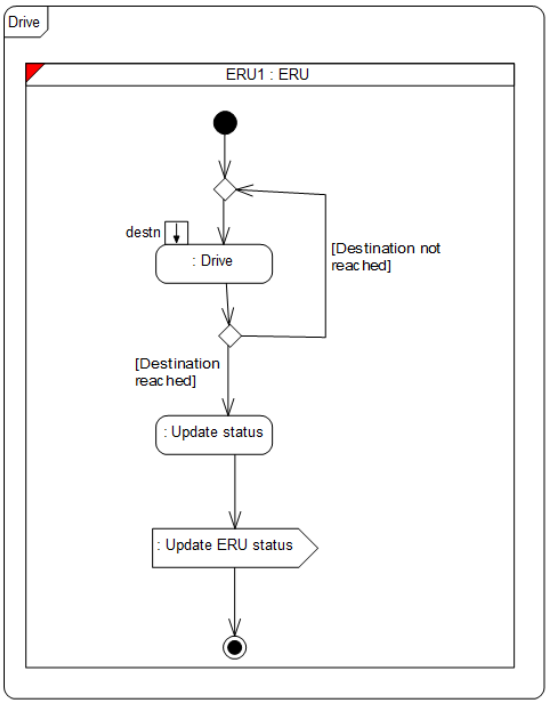

Figure 15: Processes View (drive process)

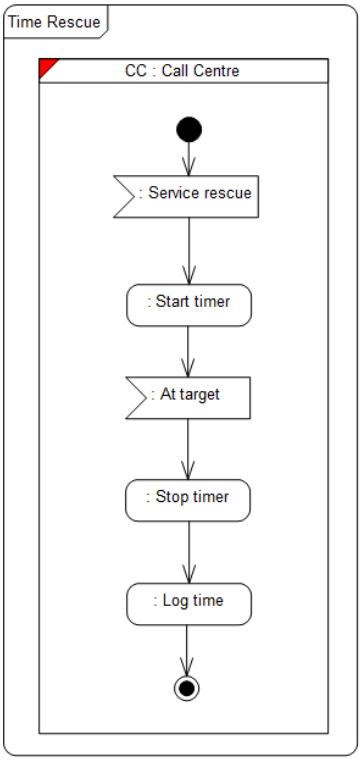

Figure 16: Processes View (time rescue process)

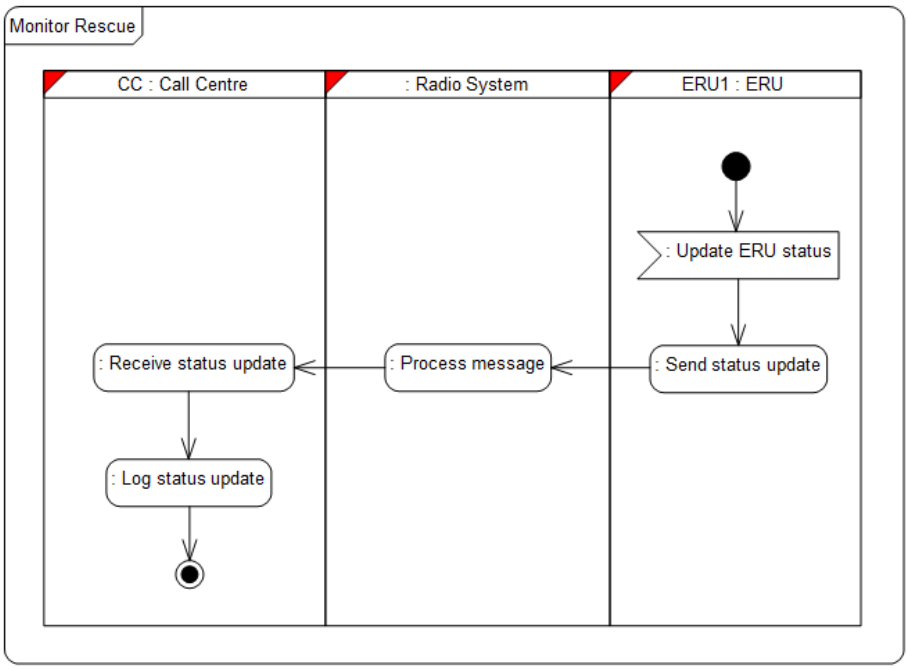

Figure 17: Processes View (monitor rescue process)

A.2 Erroneous behaviour

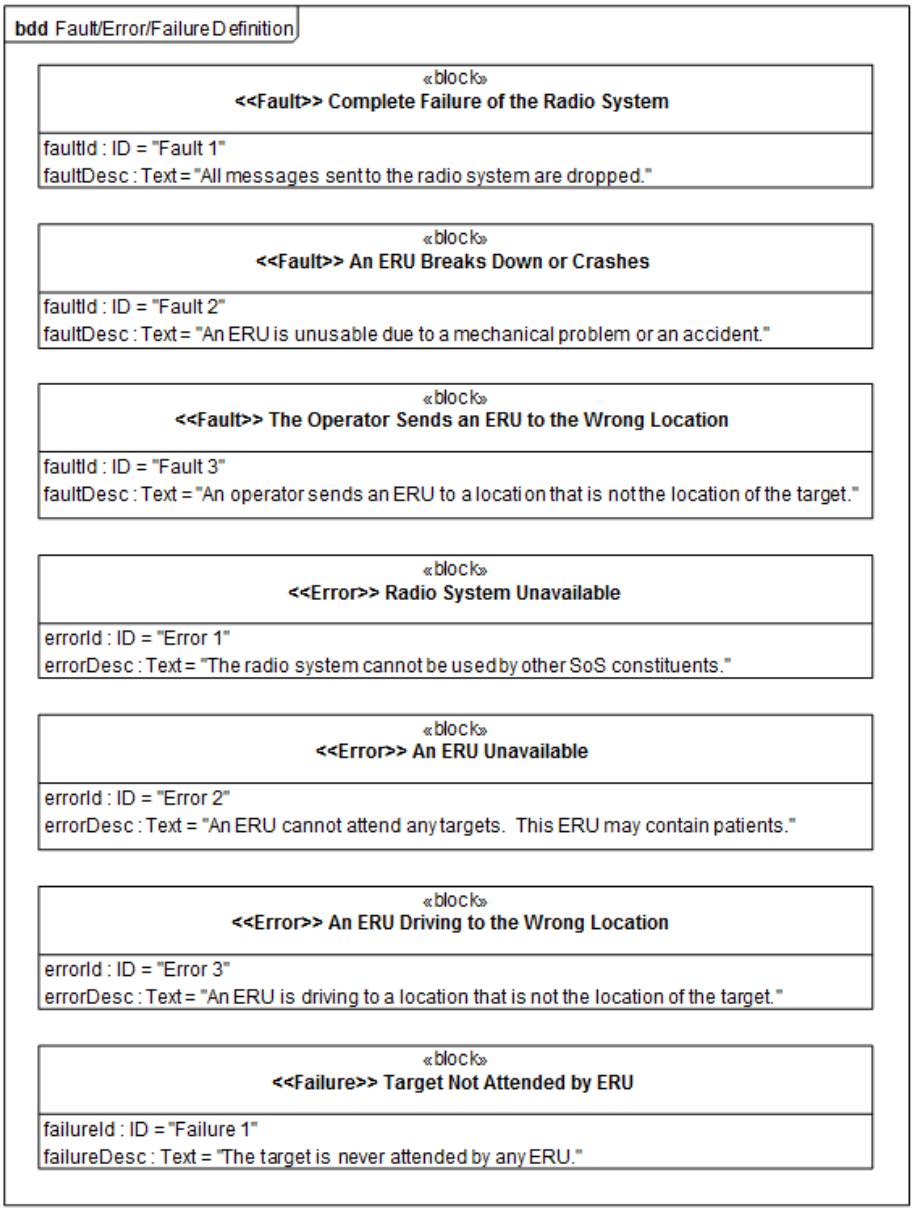

Figure 18: Fault/Error/Failure Definition View

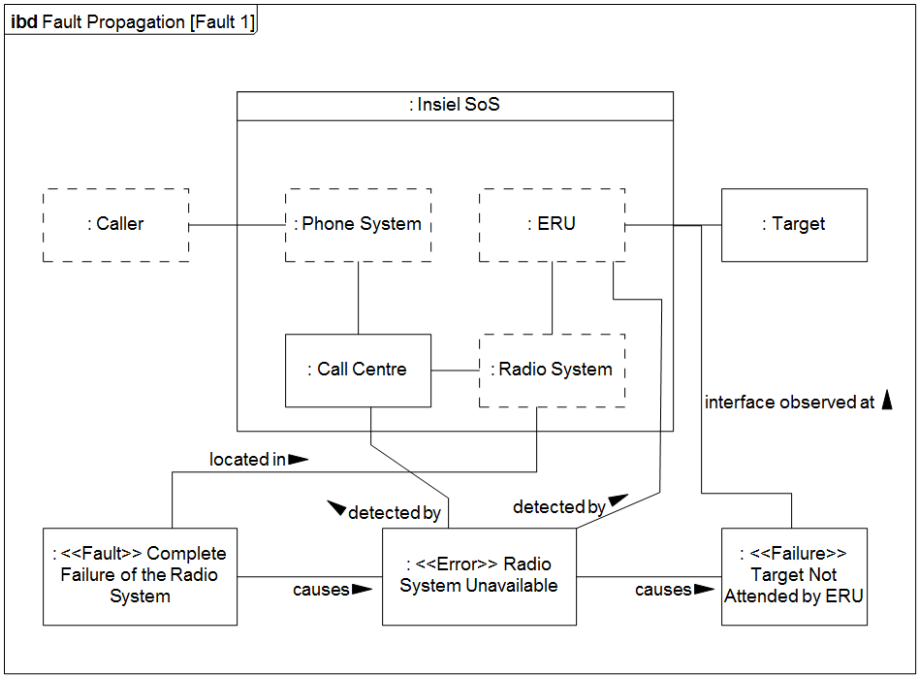

Figure 19: Fault Propagation View for Fault 1

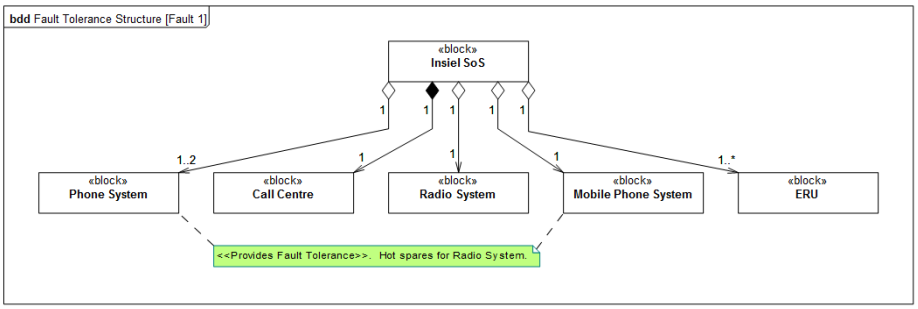

Figure 20: Fault Tolerance Structure View for Fault 1

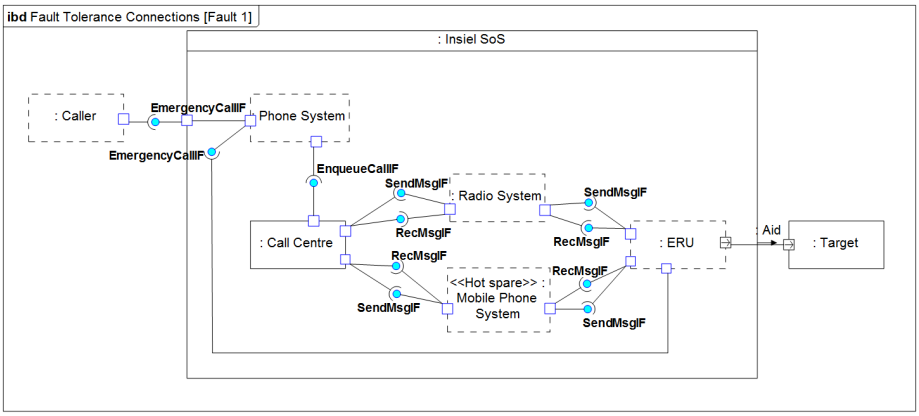

Figure 21: Fault Tolerance Connections View for Fault 1

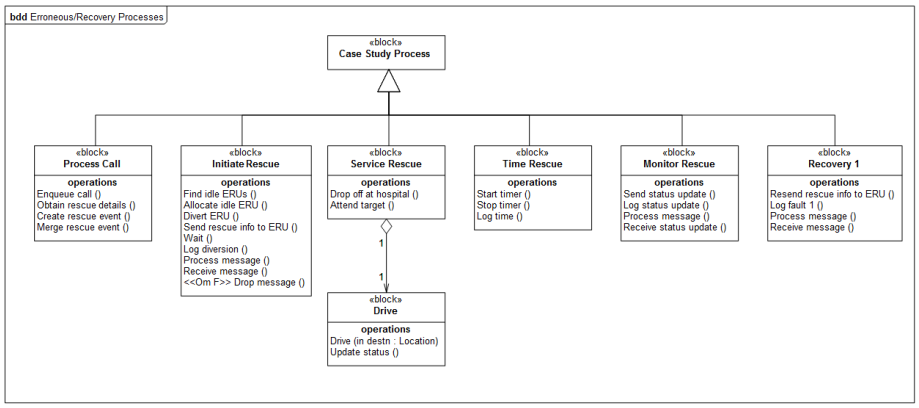

Figure 22: Erroneous/Recovery Processes View

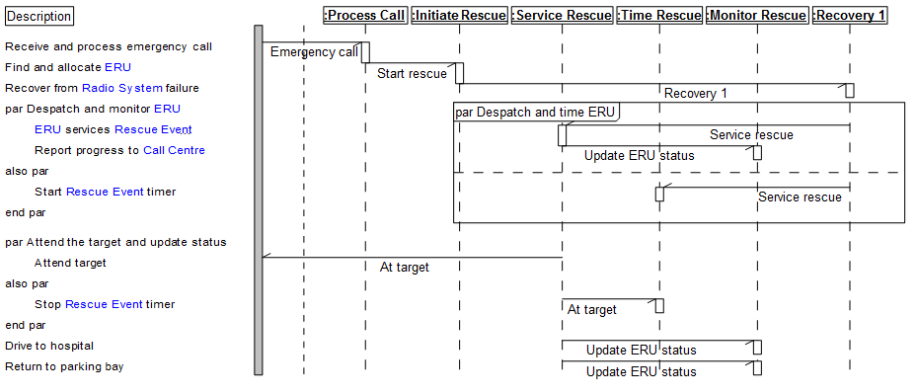

Figure 23: Recovery Scenario View for Fault 1

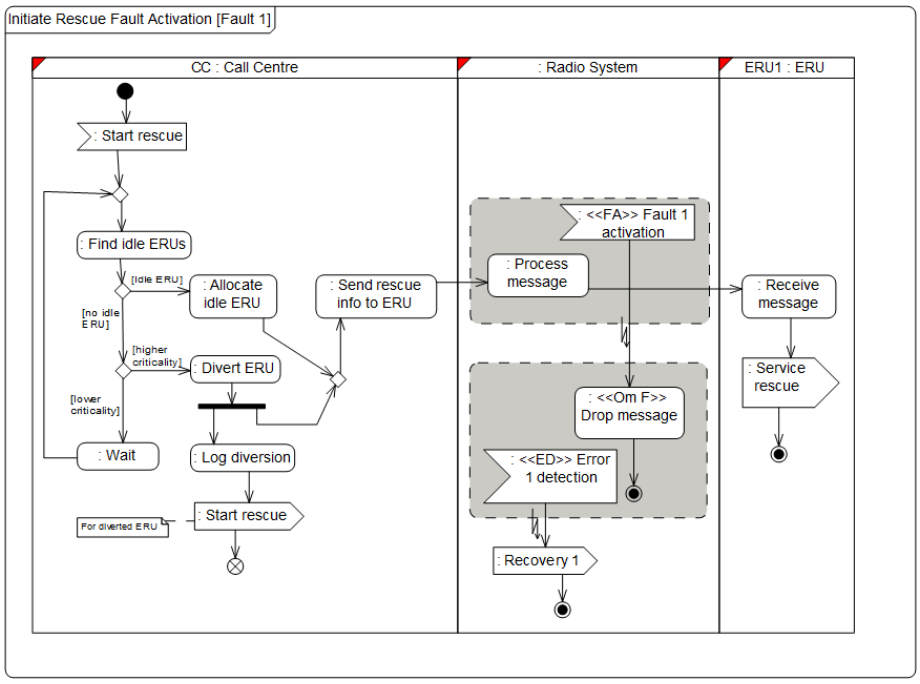

Figure 24: Fault Activation View (initiate rescue process) for Fault 1

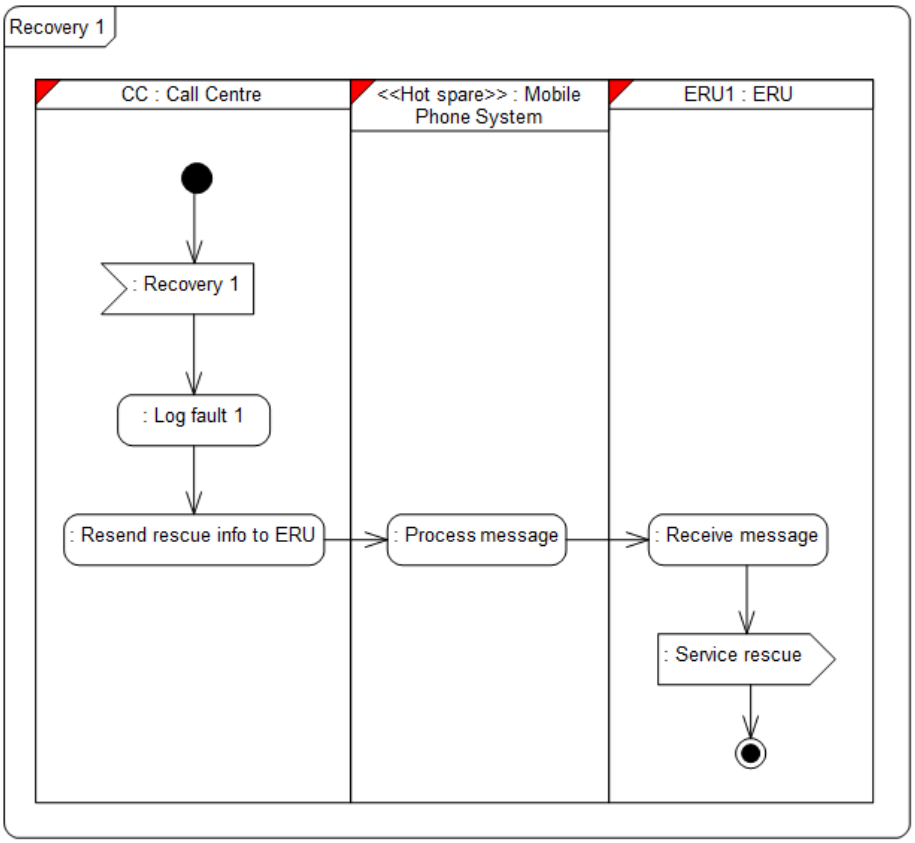

Figure 25: Recovery View for Fault 1