Contributors:
Margherita Forcolin, INSIEL
Pietro-Felice Petrucco, INSIEL
Roberto Previato, INSIEL
Richard Lloyd Stevens, INSIEL
Richard Payne, UNEW
Claire Ingram, UNEW
Zoe Andrew, UNEW

Editors:
R. Lloyd Stevens, INSIEL
M. Forcolin, INSIEL

Reviewers:
Peter Gorm Larsen, UA
Andy Galloway, UY
# Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>31.08.2012</td>
<td>P.F.Petrucco, M. Forcolin</td>
<td>Initial abstract, introduction and table of contents for review. Incomplete.</td>
</tr>
<tr>
<td>0.2</td>
<td>6.11.2012</td>
<td>P.F.Petrucco</td>
<td>Reviewed Table of Contents and initial content</td>
</tr>
<tr>
<td>0.3</td>
<td>19.12.2012</td>
<td>M. Forcolin</td>
<td>Structure revisited after the first year review</td>
</tr>
<tr>
<td>0.4</td>
<td>4.1.2013</td>
<td>M. Forcolin, P.F. Petrucco, R. Stevens, R. Payne, Z. Andrew</td>
<td>Assembled all received contributions</td>
</tr>
<tr>
<td>0.5</td>
<td>8.1.2013</td>
<td>M. Forcolin, R. Previato, R. Payne</td>
<td>Included VDM model, Partial assessment of contributions-first comments from internal peers reviewed</td>
</tr>
<tr>
<td>0.6</td>
<td>11.1.2013</td>
<td>M. Forcolin</td>
<td>Version for peer review</td>
</tr>
<tr>
<td>0.7</td>
<td>18.2.2013</td>
<td>M. Forcolin</td>
<td>Comments from 1st reviewer addressed</td>
</tr>
<tr>
<td>0.8-0.11</td>
<td>10.3.2013</td>
<td>M. Forcolin</td>
<td>Comments from 2 reviewer and NC</td>
</tr>
<tr>
<td>1.0</td>
<td>15.2.2013</td>
<td>R. Payne, C. Ingram, R. Stevens, M. Forcolin</td>
<td>Corrections and release candidate</td>
</tr>
<tr>
<td>1.4</td>
<td>10.5.2013</td>
<td>C. Ingram</td>
<td>Additions to benchmarking sections, small edits</td>
</tr>
<tr>
<td>1.5</td>
<td>20.5.2013</td>
<td>C. Ingram</td>
<td>Further edits to benchmarking sections</td>
</tr>
</tbody>
</table>
Abstract

This deliverable details the activities performed in Task 4.1.1 to analyse and model the Emergency Response System of Systems case study. This deliverable describes the use case from a functional point of view and describes the objectives that emerge. It defines the particular needs, requirements and objectives for an industrial software contractor in the development of multi-stakeholder integrated systems of systems. It outlines the particular challenges that arise in the public emergency management field that also represent interesting challenges for the system of system engineering domain. It details how this use case fits into the System of system landscape and which elements characterise it as such. The document presents an overview of the systems as-is and the main functionalities it exposes, then it will address the challenges we faced in the selection of the technologies to be used in the modelling and finally it will present the approach used in the modelling phase for the two selected technologies. The full models produced using VDM and SysML are reported in the Annexes (Annex1 and Annex 2 respectively). The Use of CML is expected to improve the systems of system engineering process by giving analysts the analytical tools and constructs to reduce the engineering process effort by codifying many of the routines which were previously impossible to represent, providing a template for processes often replicated as well as reducing the overall time dedicated to the engineering process itself. This document also serves as a description of what can be done without CML, so as we move to phase III of the COMPASS project we can investigate its applicability and demonstrate the improvement over current practice.
Table of Contents

1 Introduction ..................................................................................................................... 12
2 Benchmarking Evaluation ............................................................................................ 12
  2.1 SOS Emergency response ERSoS System .............................................................. 14
2.2 Usage Scenario ........................................................................................................... 15
3 SoS Challenges for Insiel ............................................................................................. 20
  3.1 Insiel SoS Business and Technical Challenges ....................................................... 20
    3.1.1 Legal Terms ........................................................................................................ 22
    3.1.2 Economic terms ................................................................................................. 23
    3.1.3 Political terms ..................................................................................................... 24
    3.1.4 Technical terms ................................................................................................. 25
  3.2 Engineering Challenges ............................................................................................ 26
  3.3 Case study objectives ............................................................................................... 27
  3.4 Challenge problems ................................................................................................. 27
  3.5 Emergency Response Case Study Requirements .................................................. 30
  3.6 Emergency Response SoS Outline .......................................................................... 31
    3.6.1 Emergency Response SoS Boundary ................................................................. 31
    3.6.2 Emergency Response SoS - Constituent System Functionality ....................... 34
  3.7 Emergency Response SoS Characteristics .............................................................. 35
4 Baseline Technologies .................................................................................................... 39
  4.1 Commonly adopted technologies in Insiel .............................................................. 39
  4.2 Selected technologies for Task4.1.1 ........................................................................ 40
    4.2.1 SysML ................................................................................................................ 40
    4.2.2 VDM .................................................................................................................. 40
5 Case Study – Accident Response System ...................................................................... 42
  5.1 Approach to Benchmarking .................................................................................... 42
  5.2 Approach to SysML modelling ................................................................................. 43
    5.2.1 SoS boundary ...................................................................................................... 44
    5.2.2 Requirement Classification and Catalogue ....................................................... 46
    5.2.3 Produce the overall SoS Use Case Model (SoS-UCM), .................................... 48
    5.2.4 SoS Internal Structure and Behaviour .............................................................. 49
  5.3 Challenge problem analysis in SysML ...................................................................... 60
    5.3.1 Evolution of a constituent system ...................................................................... 60
    5.3.2 A new environment ............................................................................................ 61
    5.3.3 A new constituent system which changes the architecture .............................. 62
10 SoS UCM - Sos overview ................................................................. 120
10.1 Process Emergency (CUS viewpoint) ............................................. 121
11 Block Diagrams ........................................................................... 121
   11.1 Block Definition Diagrams ....................................................... 121
       11.1.1 SoS overview Structure-baseline ........................................ 122
       11.1.2 SoS overview Structure- Fault Tolerant ................................ 122
       11.1.3 Insiel SoS key entity structure ........................................... 123
       11.1.4 Incoming Call keyword usage ............................................ 124
       11.1.5 Insiel behavioural blocks structure diagram ....................... 125
       11.1.6 Insiel SoS deployment structure ........................................... 126
       11.1.7 Call Centre "CUS" internal Structure ................................ 127
       11.1.8 Case Study Processes ....................................................... 128
   11.2 Internal Block Diagrams .......................................................... 129
       11.2.1 Case Study IBD ................................................................ 129
       11.2.2 [block] Radio System ....................................................... 130
       11.2.3 [block] Phone System ....................................................... 131
       11.2.4 [block] CUS external interfaces ........................................... 132
       11.2.5 [block] CUS Phone Answering .......................................... 134
       11.2.6 [block] CUS Scheduling ................................................. 136
       11.2.7 [block] CUS Monitoring ................................................... 138
12 BEHAVIOUR .................................................................................. 140
   12.1 Use Case Diagrams .................................................................. 140
       12.1.1 Process Emergency 1 (CUS viewpoint) ............................... 140
       12.1.2 Ask for First Aid Rescue (caller viewpoint) ......................... 141
       12.1.3 Process First Aid Rescue (caller viewpoint) ......................... 141
   12.2 Activities ................................................................................. 142
       12.2.1 Process Call .................................................................... 142
       12.2.2 Answer call .................................................................... 144
       12.2.3 Obtain casualty details ..................................................... 145
       12.2.4 Get target location info .................................................... 146
       12.2.5 Locate position ................................................................ 147
       12.2.6 Obtain further detail ......................................................... 148
       12.2.7 Integrate response event detail ......................................... 149
       12.2.8 Replay of recorded conversation ...................................... 150
       12.2.9 Initiate Response ............................................................ 151
12.2.10 Update response ................................................................. 152
12.2.11 Initiate Mission ................................................................. 153
12.2.12 Radio Send message to ERU ............................................. 154
12.2.13 Service Mission .............................................................. 155
12.2.14 Change status ............................................................... 156
12.2.15 Monitor Response ........................................................... 157
12.2.16 Time Mission ................................................................. 158

12.3 Sequence Diagrams ................................................................ 159

12.3.1 SoS level ............................................................................. 159
12.3.2 Process call ........................................................................ 159
12.3.3 update ERU status - base Monitor Response ....................... 159
12.3.4 Update ERU status - baseline (Monitor Mission) ............... 160
12.3.5 Update ERU status - FT (Monitor Mission) ....................... 161
12.3.6 Update ERU position - baseline (Monitor Mission) ........... 161
12.3.7 update ERU position - FT (Monitor Mission) ..................... 162
12.3.8 Update ERU position - improved (Monitor Mission) ......... 163
12.3.9 Internal behaviour ............................................................. 164
12.3.10 Internal behaviour using processes .................................... 165
**List of figures**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ERSoS Emergency Response System of System</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>Usage example of the ERSoS system</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>CUS Overview page and accident sinoptic</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Geographic view screen</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>CUS Operator</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>Flow of Activities performed by ERSOS</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>Emergency response SoS boundary</td>
<td>31</td>
</tr>
<tr>
<td>8</td>
<td>Baseline SoS overview</td>
<td>45</td>
</tr>
<tr>
<td>9</td>
<td>Improved SoS overview including Mobile Phone System</td>
<td>45</td>
</tr>
<tr>
<td>10</td>
<td>Requirement diagram detailing SoS functional requirements</td>
<td>46</td>
</tr>
<tr>
<td>11</td>
<td>Overall SoS Use Case Model</td>
<td>48</td>
</tr>
<tr>
<td>12</td>
<td>Constituent system external interfaces</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>Internal Block Diagram depicting CUS components and connectivity for ERU scheduling</td>
<td>51</td>
</tr>
<tr>
<td>14</td>
<td>Internal Block Diagram depicting CUS components and connectivity for phone answering functionality</td>
<td>52</td>
</tr>
<tr>
<td>15</td>
<td>Internal Block Diagram depicting CUS components for ERU monitoring functionality</td>
<td>53</td>
</tr>
<tr>
<td>16</td>
<td>Internal Block Diagram depicting CUS and Phone System components for call handling functionality</td>
<td>54</td>
</tr>
<tr>
<td>17</td>
<td>Block definition diagram containing interface definitions</td>
<td>55</td>
</tr>
<tr>
<td>18</td>
<td>Block definition diagram showing behavioural processes</td>
<td>56</td>
</tr>
<tr>
<td>19</td>
<td>Activity diagram for the &quot;Process Call&quot; behavioural process</td>
<td>57</td>
</tr>
<tr>
<td>20</td>
<td>Activity diagram showing the &quot;Obtain casualty details&quot; sub-activity</td>
<td>59</td>
</tr>
<tr>
<td>21</td>
<td>Sequence diagram depicting scenario of processing a call</td>
<td>60</td>
</tr>
<tr>
<td>22</td>
<td>Block Definition Diagram showing structure of VDM model</td>
<td>87</td>
</tr>
<tr>
<td>23</td>
<td>[Package] baseline SoS Overview</td>
<td>114</td>
</tr>
<tr>
<td>24</td>
<td>[Package] improved SoS Overview</td>
<td>115</td>
</tr>
<tr>
<td>25</td>
<td>Process Emergency (CUS viewpoint)</td>
<td>121</td>
</tr>
<tr>
<td>26</td>
<td>SoS overview Structure-baseline</td>
<td>122</td>
</tr>
<tr>
<td>27</td>
<td>SoS overview Structure- Fault Tolerant</td>
<td>122</td>
</tr>
<tr>
<td>28</td>
<td>Insiel SoS key entity structure</td>
<td>123</td>
</tr>
<tr>
<td>29</td>
<td>Incoming Call keyword usage</td>
<td>124</td>
</tr>
</tbody>
</table>
Figure 30 Insiel behavioural blocks structure diagram .................................................. 125
Figure 31 Insiel SoS deployment structure ..................................................................... 126
Figure 32 Call Centre "CUS" internal Structure ............................................................. 127
Figure 33 Case Study Processes ..................................................................................... 128
Figure 34 Case Study IBD ............................................................................................... 129
Figure 35 [block] Radio System ...................................................................................... 130
Figure 36 [block] Phone System .................................................................................... 131
Figure 37 [block] CUS external interfaces ....................................................................... 132
Figure 38 [block] CUS Phone Answering ....................................................................... 134
Figure 39 [block] CUS Scheduling .................................................................................. 136
Figure 40 [block] CUS Monitoring ................................................................................ 138
Figure 41 Process Emergency 1 (CUS viewpoint) UCD ................................................ 140
Figure 42 Ask for First Aid Rescue UCD ....................................................................... 141
Figure 43 Process First Aid Rescue UCD ...................................................................... 141
Figure 44 Process Call Activity Diagram ...................................................................... 142
Figure 45 Answer call Activity Diagram ...................................................................... 144
Figure 46 Obtain casualty details Activity Diagram ....................................................... 145
Figure 47 Get target location info Activity Diagram ....................................................... 146
Figure 48 Locate position Activity Diagram .................................................................. 147
Figure 49 Obtain further detail Activity Diagram .......................................................... 148
Figure 50 Integrate response event detail Activity Diagram ......................................... 149
Figure 51 Replay of recorded conversation Activity Diagram ....................................... 150
Figure 52 Initiate Response Activity Diagram ............................................................... 151
Figure 53 Update response Activity diagram ................................................................ 152
Figure 54 Initiate Mission Activity Diagram .................................................................. 153
Figure 55 Radio Send message to ERU Activity Diagram ............................................. 154
Figure 56 Service Mission Activity Diagram .................................................................. 155
Figure 57 Change status Activity Diagram ................................................................... 156
Figure 58 Monitor Response Activity Diagram .............................................................. 157
Figure 59 Time Mission Activity Diagram .................................................................... 158
Figure 60 SoS level Sequence Diagram ...................................................................... 159
Figure 61 Process call Sequence Diagram .................................................................... 159
Figure 62 Monitor Response Sequence Diagram ......................................................... 159
Figure 63 Update ERU status - baseline (Monitor Mission) Sequence Diagram ...... 160
Figure 64 Update ERU status - FT (Monitor Mission) Sequence Diagram .......... 161
Figure 65 Update ERU position - baseline (Monitor Mission) Sequence Diagram ... 161
Figure 66 Update ERU position - FT (Monitor Mission) Sequence Diagram ....... 162
Figure 67 Update ERU position - improved (Monitor Mission) Sequence Diagram 163
Figure 68 Internal behaviour Sequence Diagram ........................................... 164
**Figure 69 Internal behaviour using processes Sequence Diagram** .................. 165

**List of Tables**

Table 1 Characterising the Insiel case study using eight properties associated with SoSs 36
Table 2 Example of functional requirements collected 48
Table 3 Functional Requirements 116
Table 4 Non Functional Requirements 120
Table 5 Interfaces in the Radio System failure, baseline SoS 129
Table 6 Interfaces of the Phone System 131
Table 7 CUS external interfaces 133
Table 8 Interfaces of the CUS Phone Answering 135
Table 9 Interfaces in the CUS scheduling 137
Table 10 Interfaces in the CUS Monitoring 139
1 Introduction

This deliverable describes the work done and the experiences gained from studying the Emergency Response Systems of Systems (ERSoS) case study. This emergency coordination system is an operational example of a System of Systems, integrated by Insiel and used by several public authorities across the Friuli Venezia Giulia and Aosta Regions in Italy. The work demonstrates and evaluates COMPASS engineering tools in a high profile, public domain, multi-owner system of systems (SoS). This deliverable is intended to demonstrate the work done to establish a baseline of tools and methods as well as a plan to evaluate the COMPASS tools and methods that will become available during the life of the project. This work will improve Insiel’s internal engineering ability to develop, maintain and adapt SoSs effectively as well as giving guidance to other COMPASS researchers collecting requirements and developing new tools during the course of the project. In particular, as engineers in Insiel provide their requirements and advice, they will be giving a viewpoint of not necessarily what is the highest performing set of tools and methods, but rather what are the optimal set of tools and methods given the constraints of a working industrial setting with market time, budget constraints and potential competency limitations of employed software developers.

2 Benchmarking Evaluation

Throughout this deliverable results from the initial benchmarking activity for the Insiel case study are presented. A benchmarking methodology has been developed under Theme 1 and is used here to assess the effectiveness of current (baseline) engineering tools and techniques. In future deliverables these benchmarks will form the basis of a comparative assessment of tools and techniques developed by COMPASS.

The benchmarking method is based on assessing the structural and behavioural qualities of the SoS engineering processes. We identify some observable element of a tool, evaluate its effectiveness, and finally allocate an approximate percentage representing the user’s experience of the tool (we provide definitions for “effectiveness” in an SoS tool in D11.2).

In this deliverable we evaluate the VDM modelling approach using the following benchmarks (see COMPASS deliverable D11.2 for an explanation of benchmarks):

- **SoS architectural modelling (SDAA)**
  The modelling language must provide systems of systems modelling capabilities. Models must be representative of the communication channels used.

- **SoS model verification (SDVA)**
  The modelling language must provide modelling capabilities for simulation of use-cases. Simulation results must give indications as to
where interruption occurs

- **SoS Model Verification (SMVA)**
  The modelling methods must provide guidelines and modelling capabilities for simulation of use-cases, and sound SoS reasoning

- **SoS test model development (STMA)**
  The model-based test method must provide guidelines, architectural patterns and modelling capabilities for development of SoS test models

- **SoS Model Identification (SMIA)**
  The model-based method and SysML must provide aggregation and modelling capabilities allowing identification and incorporation of SoS elements and architectural patterns.

- **SoS Model Development (SMDA)**
  SysML must provide guidelines, architectural patterns and modelling capabilities for development of SoS models.

- **SoS Emergent Behaviours Requirement (EBRA)**
  The model-based method must support development of emergent behaviours requirement that conform to standard requirements quality attributes.

- **SoS Model Verification (SDVA)**
  Modelling language must provide modelling capabilities for simulation of use-cases. Simulation results must be decidable.

- **SoS Model Verification (SMVA)**
  The model-based method and modelling language must provide guidelines and modelling capabilities for simulation of use-cases, and sound SoS reasoning.

- **SoS state transition relation (STRA)**
  The model-based test method must support modelling of states and transitions for both the constituent systems level and the SoS level. The transition systems of the test models must be verifiable

- **SoS Model Terminology Consistency (SMTA)**
  The semantic strength of the model-based method’s terminology must support rule of inference logic through SoS modelling concept definitions and guidelines, and thereby eliminate ambiguous communication and modelling among domains.

- **SoS Change Propagation Analysis (SCPA)**
  The model-based method should provide excellent support for identifying dependencies between constituent systems and the likely candidates that
will be affected if changes are introduced by a constituent or the environment

- **SoS Interfaces Analysis (SIAA)**
  The model-based method should provide excellent support and guidelines for the identification and analysis of interfaces and contracts both internally and externally

- **SoS Fault Tolerance Analysis (SFTA)**
  The model-based method should provide excellent support and guidance of the analysis of fault tolerance, by helping to identify possible faults, events that trigger them and supporting the development of recovery strategies

The percentage awarded is transformed into a single integer, as follows:

- 0 = 0%
- 1 = 1-25%
- 2 = 25-50%
- 3 = 50-75%
- 4 = 75-100%

The benchmarking process defined in D11.2 describes these measurements and how they correspond to the AAA classification system described in D11.1 and D11.2 based on the output of the evaluation method.

The concrete workflow of the benchmarking/evaluation method for this deliverable consists of modelling aspects of the Insiel’s Emergency Response System SoS using a baseline technology. Observable concepts are evaluated for the ease of modelling an element and effectiveness of the resulting model. The observable concepts are validated against expectations regarding SoS challenges established already in Deliverable D11.1 and Deliverable D11.2. The observable concepts are:

- Methods for SoS development, including guidelines, analysis and outlines
- Architectural modelling for SoS, including identifying patterns for modelling verification
- Analysis techniques for SoS, including compositional techniques and fault tolerance modelling
- Simulation including testing capabilities for SoS models.

Results of the benchmarking evaluation are presented in Section 5.10.

### 2.1 SOS Emergency response ERSoS System

The Emergency Response System of Systems integrates existing emergency response systems from hospital emergency room management software, ambulance and helicopter dispatch services, civil protection management systems, mountain rescue and fire brigade dispatch systems.
The ERSoS system acts as a supervisor and coordination system in the case of large-scale emergencies that require coordination of independent emergency response systems. The SoS also interfaces with other institutional exchange systems, including local police and hospitals, which are not tightly coupled with the emergency services. Although these systems are all developed, managed and maintained independently, the ERSoS has been implemented to allow the systems to share a common workflow and share resources as a pool. The systems recognise an event and, based on predefined rules, delegate control to any actor in the ERSoS, which will then act as a centralized synchronization unit for:

- Coordination;
- Command;
- Resource Optimisation;
- Communication;
- Monitoring;
- Reporting

of shared resources distributed throughout the territory in question.

**Usage Scenario**

The SoS is in constant use by local participating authorities. In normal operation, it can simply be considered a good resource coordination mechanism, providing a higher level of accountability in situations where resources are shared. However, in cases where a high level of rapid shared decision-making is required, the efficacy of the system is more apparent. This is demonstrated in the following scenario, introduced in Figure 2.
A cargo train moving along a mountainous valley in the Friuli Region derails. As the cars pile up at the derailment point, several of the crew and rescuers are seriously injured and will eventually require helicopter evacuation to the nearest trauma center, which is over 80 kilometres away. One of the train cars spills over into a nearby highway, injuring a driver and passengers who will eventually require triage and emergency evacuation.

The impact has started a fire away in the trees and brush which will require fire brigades with particular equipment. Several of the tanker cars contain flammable vinyl chloride, and one tanker releases liquid, which vaporises into the surrounding air and may be leaking into a nearby river. Several people in the vicinity will later complain of respiratory problems. A number will need to be evacuated to nearby medical facilities and residents in the immediate area will need to be evacuated. Symptoms of sore throats will
prompt authorities to send rescue workers and others who have been exposed to local hospitals for decontamination and treatment.

In total, 2 seriously injured crew members and one fire fighter are brought by helicopter to a regional unit and 11 people with light injuries or breathing problems are taken to local hospitals. 17 walk-in patients with respiratory problems will also visit the local emergency rooms. During this scenario, the healthcare emergency response unit received a 118 call reporting the incident from an uninjured witness on the highway. This unit manages emergency vehicles and the interface to hospital emergency rooms. The phone operator taking the call opens an accident form on the CUS system, the first stage of the CUS system management. The caller mentions seeing open flames, so, following protocol, the operator contacts the fire brigade and the system passes “control” of the form and event to the fire brigade operator. Because the call is received from a cell phone the operator asks for incident location information (subscriber geo-referenced data is collected from the telephone exchange geo-referenced database if the caller uses a fixed line). The cartography system maps the incident location to the SITER regional cartography data and the location comes up on second screen along with data on emergency vehicles, routes, event locations and resources like hospitals and emergency vehicles spread across the territory. The operator must decide which response units to inform and activate and which additional systems must be informed of the incident. Following selection by the operator, the system automatically sends messages to the local police with details and location of the incident. The operator can also speak directly with these services’ management. The operator now sends the first ambulances and fire trucks to the scene, which produce further data on the injured and the required resources as they arrive. The human CUS operator involves the civil protection department and a forest department. The ambulance itself has contacted the emergency triage system from the emergency room, which is routed through the CUS to the regional emergency radio system.
A civil protection official on the scene is relaying information through the ERSoS to evacuate nearby residents. The operator monitors this, deciding to allocate additional ambulances and also to dispatch the emergency helicopter from the Udine Hospital.

The CUS operator can maintain contact with any of the individual control centers or emergency vehicles, but more often will patch emergency operators to one another to manage particular facets of the incident. Figure 6 shows the flow of events during the course of an emergency from the time a call is received until the resources are dispatched.

Figure 6 Flow of Activities performed by ERSoS

In the situation described here a single operator has been given authorisation by the constituent system owners to:

- Define an event in their local systems
- Allocate responsibility to one or more emergency operators
- Determine what is the most advantageous type of response to the conditions that have arisen
- Plan an appropriate response
- Physically allocate and dispatch resources managed by any individual constituent systems.

Control and dispatch of the shared resources and a real-time overview of any situation is obviously the most desirable part of the system, but interaction with the constituent systems requires a high level of system interaction. The constituent systems all understand and allow a number of functions to be initiated centrally from the ERSoS, and also allow the overriding of their prime functionalities in many cases. An approach based on Business Process Management and web service-based system communication has been adopted in order to perform the number of “activities” really required to allow the system collaboration.

From a functionality point of view simply allowing event time-sharing of resources and system interaction would not be enough to ensure system level integration. The ERSoS performs a number of required coordination, monitoring and reporting functions to the various constituent system owners including:
a) Radio and telephone interface management
b) Cartographic service interface
c) Call reception management
d) Intervention / mission management
e) Protocol management
f) Vehicle and equipment management
g) Secondary vehicle management
h) Personnel management
j) Shift management
k) Number mapping and agendas
l) Other emergency system integration (Medical Priority Dispatch System (MPDS), SIGEM-SIMMA,...)
m) Statistics, reporting functions
n) Integration to regional public health registry office
o) Interfaces to fax server, SMS gateway
p) Resource location System (GPS and cell location)

Some of the primary services like dispatching an ambulance may seem essential but in fact some of the secondary services like “j” shift management will condition the availability of the primary service. It is exactly these functionalities that lie embedded in the constituent systems that demonstrate the level of complexity and interaction within this system of systems.
3 SoS Challenges for Insiel

3.1 Insiel SoS Business and Technical Challenges

Insiel is a natural actor to plan this particular type of system of systems, and is responsible for producing the CUS constituent system of the ERSoS. Insiel operates on the software system market as a private company, which is however 100% publicly owned by the regional government, the municipalities, and the health care authorities. These actors are the levels of government that are responsible in an emergency event which span across jurisdictions, levels of government and authority lines. Where these actors would struggle to collaborate, Insiel offers a neutral entity to facilitate collaboration between all stakeholders. Additionally Insiel has developed software used to realize the constituent systems for these government actors. Insiel has competence in these areas and offers interoperability across a wide array of public systems, covering a variety of domains from city management functions to health care management and including wide area geographic information systems. These systems are for the most part independent and carry out a necessary function without any interaction from external systems. However, they are increasingly expected to provide functionalities for reporting and data management across systems and across levels of government. As the market evolves the typical systems are increasingly expected to interoperate. Longstanding municipal applications like citizen registries are increasingly being used to provide access and contribute towards record keeping in large hospital systems. Although independent and autonomous, they are expected to interoperate and work as an SoS, fitting together like puzzle pieces. In the public domain the concept of SoS engineering is becoming ever more important.

Providing integrated software for public entities in periods of financial pressure is challenging. Administrations insist on comprehensive solutions with evermore demanding levels of quality at reduced costs, in shorter time frames. At the same time public organisations are dynamic with shifting organisational units, changing officials and jurisdictions, which has a profound effect on software development requirements even in on-going projects. Managing information about physical persons and, in this case, caring for their physical well-being, adds extensive security and dependability requirements, not only for the software developed but also on the communications networks and data storage. Legal constraints that most software can eschew can invalidate development creating new requirements. In the particular ERSoS case study all of these problems are constantly changing the development landscape. The case study is particularly challenging due to the fact that the resources managed, on one hand require a very high fault tolerance and on the other, are frequently changing.

There is a requirement for a high degree of fault tolerance due to the fact that if the ERSoS is expected manage a resource there can be no equivocal consignment, no error in its use and there must be exact communication with all operators and systems involved. The manager of the ERSoS takes the legal and social responsibility for the wellbeing of physical people. Emergency vehicles,
communication and medical and rescue support systems, as well as hospital and emergency room systems are registered and enrolled by the ERSoS. As this occurs the system must ensure that protocols, units and measures and database requirements from different systems are understood and correctly set and can be managed and monitored. The system changes and reconfigures frequently because it is essentially a resource management system using business intelligence to manage resources from various systems. Resources can be people (i.e. ambulance drivers, triage medics, doctors etc.), equipment or vehicles (pacemakers, ambulances, rescue equipment, etc.) that have lifespans and may move in or be retired from the system. The system as mentioned is itself made up of a number of constituent systems with “owners” that may be in different administrative jurisdictions. These owners are autonomous in their choice to update equipment and software, creating further need to update.

The low tolerance for errors, tied to the constant need to review and examine the impact of new constituent elements, creates the need for frequent engineering review. To facilitate identification of interoperability and control elements of constituent systems, the CUS has adopted a Service Oriented Architecture (SOA) approach, where constituent systems expose standardised XML metadata and system functionalities as WSDL. These services are exposed to the CUS in a manner as loosely coupled as possible, to reduce the integration and interoperability concerns. Nonetheless, given the nature of the system, where independent operational dispatch and command centres relinquish control of their resources in crisis situations, there is commonly the need to interact with the constituent systems by the master system as resources are appropriated and registered. Primarily this is due to the high fault tolerance requirements, which demand that there is always a backup architecture available. When the master system or any of the communications networks connecting it should become unavailable, the backup control system in the backup server farm is actuated but control can be transferred to any of the constituent system control centres or even to a mobile control center, perhaps in the geographic proximity to the crisis event. Any of these can be automatically reconfigured to become the master system. For this purpose there is commonly the need for the master system to know the status of all backup systems that are available (individual command centres included). Additionally geographically distributed control centres, which are satellites, may need to update information regarding a particular resource. These factors limit the ability to implement very loosely coupled services in the SOA architecture. In CUS systems provided by Insiel, new actors, new resources and new resource types do appear and others are decommissioned. Reconfiguration and completely new constituent software is deployed. As the system is not truly loosely coupled, the impact of changes (in functionalities or resources) in the constituent systems needs to be fully examined to ensure that the data provided or required by the constituent systems will create no conflict with other elements in the system.

Considering that the CUS system architecture has been developed and systems have been implemented, combined with the fact that new systems are essentially a replication of existing systems, software engineers working on these systems
currently spend the majority of their time analysing the interaction and impact of changes within running systems. Currently each interaction is traced through the various relationships through the entire lifecycle of the information consumed or generated. Testing carried out on each separate system creates excessive redundancy. Security constraints, components, activity, behaviour, use case and entity relationships are, in fact, all documented however they are often formalised in different formats and using different tools. These documents are used to test flows of information but contain few formal models representing the impact and boundaries of the interaction. Testing and analysis are often case specific and little documentation and models are reused from previous testing. It is expected that a systematic model based approach to analysing the interactions can provide a reusable starting point and provide a significant reduction of the time required to certify constituent system “compatibility”. In economic terms conformance analysis and testing represents the single largest maintenance cost of the system. It is expected that adopting a SysML+CML model-based approach and formalising the procedure required for conformance testing will not only reduce physical time required to carry out changes impact analysis but will also highlight reoccurring constraints or bottlenecks, both in terms of systems data and communications as well as physical resources that are generating changes in the system. This will lead to an improved process planning and improvement of the systems themselves. Systematic modelling will allow process comparison, helping analysts to determine optimal process flows. It will also highlight where a particular process affects the critical path of service delivery or where, vice versa, the process or activity performed by the system has little impact on running operation during an emergency. In this manner additional monitoring can be focused on key processes. Furthermore model-based engineering will help us to understand the ramifications of updating software or adding / removing a system or a resource on a single constituent system. This is expected to facilitate the identification of processes that could potentially use common resources or that might be developed concurrently. This will help not only in planning the resources and implementing a project plan, but a priori it will help budgeters provide more accurate quotes for providing and maintaining new software installations before new systems are installed.

3.1.1 Legal Terms

Participation as a COMPASS case study is particularly important for a company like Insiel in this type of market. Knowledge and practical competence in Systems of System engineering is important for legal, technological, economic and political motives. In legal terms developments in the tendering for large government software projects require extensive system modelling. Starting in the defence industry in the United States, obligatory integrated system modelling in the tendering phase is quickly moving throughout government sectors and has spread through Canada to the UK and now is starting to be felt across Europe. Insiel participates in these types of tenders and is increasingly finding the legislative prerequisite to provide enterprise class systems models showing dependencies and interactions, and depicting evolution and cross system
security and authentication. Insiel frequently has to provide evermore-complete systems models directly in the tender phases themselves. In legal terms enterprise class SoS modelling also protect systems developers like Insiel from legal retort, demonstrating that appropriate forethought and system testing was carried out and consideration of where systems were subject to disruption had not been neglected. Integrated systems modelling is considered an appropriate method of demonstrating prudence and forethought above and beyond the functioning of the system for which it was tendered, but moreover fitting into the technological framework of the deployment scenario expected in the tender. Additionally the use of comprehensive system modelling is expected to facilitate changing legal rules regarding provision of emergency services. The legal framework regarding the response to emergency situations is in constant evolution and is often a system level obligation. For example a system component may be required to respond in a set amount of time, or the system obtaining a formal authorisation token to act on behalf of another agency will require reprogramming or parameterisation of a component. Having COMPASS modelling and testing capabilities will decrease the effort required to analyse and enact these frequent changes. Models or portions of models can be reused.

The formal semantics of COMPASS also provide a well-defined language that government procurement agencies can use to communicate and verify their requirements, while Insiel can use the same to contractualise and limit development to those elements formally expressed, saving unnecessary or unexpected development. The same formal semantics create record of agreement and contractual obligations for future reference, and document the requirements expressed by the contracting agencies.

3.1.2 Economic terms

In economic terms SoS Engineering is important for Insiel due to the fact that competition for the development of public domain software is becoming more aggressive. Contracts cannot be pre-allocated or reserved for owned in-house, publically owned companies like Insiel. They must also compete and win public contracts with private systems developers. In fact, legislation guarantees fair market accessibility and competition to all market players. In the case of the emergency response system, Insiel has in-depth commercial and technological knowledge of the constituent systems and a long-term investment (because they developed a number of the systems). In the event that the contract to develop the next generation SoS coordination system is won by a “competitor”, the public authorities would require Insiel to relinquish knowhow, interfaces and libraries regarding the particular systems that will be included in the SoS to the winning contractor. Inversely, if Insiel were chosen to expand the next generation system, Insiel would acquire direct product knowledge and market share from competitors whose systems were to be integrated. In the next generation SoS integrated software development there may be organisations whose software was not developed by Insiel. The contract winner not only gets insight into the inner workings of competitor’s software, but also has a foot in the door of that software vendor’s client, whose system is to be integrated. Therefore in these
types of government contracts for multi-component/systems bidding, SoS engineering knowledge is the key to demonstrating that a supplier is able to deliver integrated reliable systems. Whoever is able to document and describe the system interactions and global functionality will receive a higher technical score in the bidding process and, all else equal, objectively have a technological advantage.

Bidding on public complex service software often involves incorporating a significant amount of leeway for overspending due to unpredicted development. SoS engineering promises to lower the cost of examining and providing alternatives. The contractor that is able to satisfy the requirements with the lowest cost of system component development will have a competitive advantage. With a complex system, reducing leeway through more assured development costs provides this advantage. Engineering from an SoS perspective will ensure that components developed to integrate constituent systems are thoroughly analysed and the risk associated to choices is significantly abated.

Once contracts are assigned and projects developed, the cost of bidding and executing similar jobs decreases, and SoS engineering competence will help developers to provide reusable modelling components, validating and testing regimes to provide software.

Efficiency should be realised through deploying and maintaining systems of systems. In particular COMPASS tools should help speed up validating and testing newly inserted system components and upgrades, where similar components have been modelled previously. Great efficiencies can be achieved replicating systems based on existing modelling performed for previous projects.

COMPASS-based testing of systems or components before deployment will also offer economic advantages as simulations can be done in the model rather than having to rely on parallel test systems, as is typically the case for emergency response systems. COMPASS will give us the possibility to plan, model and run/simulate large number of test cases, including unit testing and big bang techniques, changing a high number of variables. With this increased testing capability, activities like the planned or cyclical updates of constituent systems (which is quite common in the system) become less expensive. Globally speaking COMPASS will help realise economies of scale, where through using COMPASS tools, companies like Insiel can move from typically unique, one-of-a-kind software development projects to a standardised, component-based product development by reusing resources from one project to the next.

### 3.1.3 Political terms

In political terms SoS knowledge is an important skill for an integrated systems developer like Insiel. A system like the ERSoS is obviously based on the autonomous systems of the organisations to which they belong. Although it is logical to build and share one ERSoS, in fact it is utopian and beyond the current mind set and normative practice of the single “owner” organisations. At a high level the organisations have a desire to maintain the responsibility and budget control over their systems. Systems management and IT department personnel
already exist in each of the organisations, and additionally they may operate on different levels of government (local/regional/national) and cannot be expected to share development and maintenance costs for the ERSoS with other levels of government, or even other departments within their organisation. Additionally the IT directors in the individual organisations fear for their personnel and their own positions where integration projects are discussed. Finally these organisations do have typical and unique software processes, which in many cases are running critical software. For these reasons no real central control over the decision process of the state police, civil protection, and hospital authority exists or is desired. Although the different actors may actually be part owners of Insiel, when one of the constituent system owners autonomously chooses to make adaptations, upgrades or migrate to new versions of software there is little discussion with Insiel beforehand. The burden to “re-integrate” and ensure compatibility falls on Insiel (the SoS supplier). These choices are usually presented when the constituent system owner has already taken the decision. In these cases SoS engineering can present simulation and testing methods that greatly increase the ability to model the systems and run extensive what-if scenarios to test a large number of loads and conditions hypothetically without preparing physical test systems. This, of course translates into time and money, and ultimately again a competitive advantage for a system developer like Insiel. However leveraging SoS engineering and the COMPASS tools, we can expect to demonstrate savings and efficiency gains to these independent actors, demonstrating how they can maintain autonomous system planning and lifespan while documenting savings they can achieve sharing a number of common resources.

In political terms the modelling and presentation approaches presented by the COMPASS tools, which in many cases are based on industry standards, become a neutral terrain between constituent systems owners, depicting the “individual systems” with a common representation and semantics. Fostering a discussion on a common model is expected to facilitate planning updates and modifications within the “community”. If this is achieved it allows discussion and planning of synchronised updates for constituent systems, which allows for greater efficiency by providing a documented mechanism of estimating impact and managing changes in each constituent system. COMPASS-based models will help describe proposed development, existing architectures, discuss changes, and communicate intentions clearly.

### 3.1.4 Technical terms

From a technical point of view, SoS engineering improves the quality of the code developed, reducing the number of errors, conflicts and failures, all of which generate high maintenance costs. Having a detailed model of cross systems interactions will allow us to anticipate interoperability concerns when new components are introduced in an independent constituent system. It will also help us inform the constituent system owners of the potential results of updates and component substitutions. An SoS level description can act as a documentation mechanism to select SoS compliant components, and to
orchestrate their introduction into the system. Constituent system owners do not choose to disregard interoperability or to introduce elements that conflict with other constituent systems, it is simply that without COMPASS-like tools it is very difficult to document and communicate compliant approaches. Constituent system owners want advice when implementing their own updates, because the ERSoS is paid for by the owners of the constituent systems. Additionally modelling and SoS-level testing will facilitate the identification of bottlenecks and potential system failure points. This is important because constituent systems in this domain have requirements for high availability that end up creating several redundant components at the constituent systems level, multiplied by the number of constituent systems. The COMPASS approach could help identify critical areas and ensure reliability coming from the redundancy of backup infrastructure as part of other constituent systems rather than having to ensure redundancy of critical infrastructure at single system level.

Using a system-to-system analysis to determine conflicts becomes difficult where a number of constituent systems rise. Predicting conflicts where changes do occur without a complete SoS model results in overlooking essential problems. Changes may include scoping areas where a whole new subsystem is introduced, to updating an existing system, or may appear minor, like a small change in the rules or workflow of a constituent application. Understanding the reach and implications of changes is complex without holistic models. If an interaction is overlooked it may block access to a system or cause an unexpected malfunction in a neighbouring system. Insel expects to gain competence on developing and maintaining comprehensive SoS models and expects them to greatly improve conflict prediction and testing.

COMPASS tools are expected to help model usage as well. Currently systems are developed to respond to any single constituent’s maximum usage scenarios, while in fact analysing scenario models might lead to improvement in handling simultaneous emergency events while optimising the infrastructural ICT elements at the base of the systems.

### 3.2 Engineering Challenges

At the beginning of the project Insel has little experience of SoS modelling as it is percieved by the state of the art in the research community. Although single groups have adopted system modelling approaches it has largely been restricted to depicting the components to be developed and describing interfaces and requirements of single constituent systems. However by modelling and testing entire systems with an SoS approach we believe we can achieve the benefits described in the earlier sections. It is important for Insel to delimit these benefits in practical terms if the engineering culture is going to be changed. To clarify the benefits we have described firstly the objectives of the project for Insel, and secondly a collection of “challenge problems” which will be tackled during the course of the Insel COMPASS case study. The challenge problems are drawn from actual problems that we can currently observe in potential business
processes and scenarios from the Emergency Response Stakeholders, and will give us the possibility to focus on Insiel’s specific objectives.

### 3.3 Case study objectives

Below we introduce the objectives of the Insiel case study.

1. **Improve the ability to implement changes across the SoS.** Changes may arise from evolution of reconfiguration. Insiel places an emphasis on change impact analysis (e.g., being able to trace the constituent systems that will be affected by a requirement change) as well as improvement of the subsequent implementation and testing. An improvement in this area will help to reduce development and maintenance costs.

2. **Develop techniques appropriate for the analysis of SoS-level properties, particularly SoS-level fault tolerance, security and emergent behaviour.** This should result in an SoS that is better able to deliver on non-functional requirements (NFRs), which have a crucial importance in Insiel’s safety critical domain.

3. **Improve the ability to analyse and optimise SoS architecture and behaviour.** For example, this would be helpful for identifying where bottlenecks arise in delivery of SoS functionality. This would help Insiel to develop strategies to improve SoS performance and efficiency, or to evaluate the possible advantages and disadvantages of different architectural configurations. An improvement in this area facilitates a proactive management of the SoS, and would also increase Insiel’s ability to adapt the system to different environments or requirements, through architectural and behavioural analysis.

4. **Make external dependencies of the SoS more visible.** This will mean that constituent systems at greater risk of unexpected changes propagated from the environment can be identified. Protective steps may be taken to reduce the negative impact that unexpected changes can have (e.g., spending more time on traceability or documentation for high risk areas, or negotiating with external entities to provide advance warning of changes). This would allow the impact on SoS performance to be managed and avoid crises where system performance is unexpectedly degraded because an external party has deployed some change without warning.

### 3.4 Challenge problems

In this section we outline some challenge problems drawn from Insiel’s real world business domain, which will be analysed using baseline technologies and, in future deliverables, COMPASS technologies. We do not analyse all four challenge problems in this deliverable, but we expect to return to these challenge problems repeatedly in future deliverables. The challenge problems have been designed to ensure that all of the objectives from the preceding section will be
addressed. We present them here along with some key questions that each problem prompts us to answer.

A. **Load testing**
   Different organisations involved in the SOS decide to split up the cost of running the system. An agreement is made to base the agreement on relative utility.
   
   1. How can the SOS model help understand how the system load is being used?
   
   2. How can an integrated load testing be set up to measure load across different systems?
   
   3. How can reporting functions support this process?

B. **Evolution of a constituent system**
   A major consideration for the ERSoS is the issue of dealing with constituent system changes. There are many examples – for instance, consider the following scenario: all constituent systems integrated into the ERSoS can be queried for questions like resource or staff availability. One of the constituent systems belongs to a public network that changes its Federated security from the OAuth 1.0 Protocol to the OAuth 2.0 Protocol. Alternatively, emergency response units (ERUs) are also considered to be a constituent system in the ERSoS. Currently ERUs report their location to the operator manually by radio, and the operator enters this on the system. However, we install equipment that allows ERUs to broadcast their information automatically to the CUS. Considering a scenario like these where a constituent system changes its interfaces:
   
   1. What services and functionalities of the ERSoS system will be affected?
   
   2. Where exactly will this impact the ERSoS System?
   
   3. What code will need to be changed?
   
   4. How can a test scenario be performed to simulate this change before it is made?
   
   5. Does it affect the level of security across the SoS as a whole?

   This challenge allows us to evaluate Objective 1, which concerns change management, as well as Objective 2, which concerns the analysis of SoS-level properties (in this case, security).

C. **A different environment**
   Insiel wins a bid to carry out another installation project in a neighbouring region, and must now estimate the cost of integrating many identical systems in a different environment.
   
   1. How to test if interfaces are different from the implementations in the existing installations in Aosta or Friuli Venezia Giulia?
2. How to understand how much of the system model is replicated where it differs?

3. How to map the SoS model to the economic effort?

4. What differences are there between the new environment and the original environment of the SoS as it is installed in Friuli Venezia? Are there any new environmental dependencies introduced?

5. What is the optimal architectural design for the new region’s SoS, and does it need to change from the architecture employed in Friuli Venezia or Aosta?

6. Are there likely to be any bottlenecks or weaknesses (in regards to fault tolerance) in the new system, and if so where will they be?

7. How to demonstrate that the new region’s system delivers the same SoS-level properties (e.g., the same standard of fault tolerance or security) as the original SoS installed in Friuli Venezia.

This challenge allows us to evaluate Objective 1, which concerns change management; Objective 3, which concerns analysis of different architectural configurations; and Objective 4, which concerns identification of external dependencies (i.e., dependencies between the SoS and its environment).

D. **A new constituent system which changes the architecture**

A new constituent system is added, which changes the way in which the constituent systems communicate with the rest of the SoS. Alternatively, a single constituent system evolves and changes how it interacts with its peers. In a hypothetical example, two of the constituent systems (ambulance service and fire service) may decide to merge their helicopter provision, instead of managing their helicopter capabilities separately. They install and share a new constituent system for allocating the shared helicopter resources between the two services. Other examples could include: the way that ERUs communicate with the CUS is changed, so that location information can be communicated directly to the CUS without the intervention of the human CUS operator speaking to the ERU staff via the radio (the latter is the current method); and a new constituent system representing some backup radio is introduced for ERUs to carry.

1. What new interfaces are now available internally and externally within the SoS and who would be responsible for managing their introduction?

2. Are changes required for any of the other constituent systems, now that there is a new constituent system added to the SoS?

3. What is the optimal architectural topography for the SoS now? E.g., in the first example, one option is to retain a “star” centralised topography, where constituent systems (including the new helicopter scheduler) communicate with the CUS and not with each other.
Another option is to adopt an alternative topography or architectural pattern, to allow the fire and ambulance constituent systems to communicate directly with each other, or perhaps they communicate with each other only via the helicopter scheduler.

4. Does the new topography have any effect on SoS-level properties, such as fault tolerance? If so, how do we demonstrate SoS-level properties still reach an acceptable standard?

This challenge allows us to evaluate Objective 3, which is about optimising SoS architectural structure; Objective 1, which concerns change management; and Objective 2, which concerns SoS-level properties.

To achieve these objectives, we first applied the COMPASS SoS modelling approach to transform existing requirements, behaviour and actors to a logical infrastructure configuration. Models were developed based on existing methods and tools. This will provide requirements to be considered in the development of new COMPASS tools in Phase IV of the project and will allow us to compare the challenge problem compliance in the earlier phase with the later phases of the project. These models are described in the following chapters of this document, providing a technology baseline.

### 3.5 Emergency Response Case Study Requirements

Given the high-level description of the problem domain in Section 1, the outline of Insiel’s business needs in Section 3.1, and the objectives specified in Section 3.2, we describe requirements for an appropriate case study, which are given below.

- The case study must describe a realistic part of the ERSoS that Insiel is responsible for. For Insiel to gain relevant experience from the case study, it should contain elements for which Insiel is an expert. This will ground the case study in realism by, for example, including real requirements and avoiding inappropriate assumptions.
- The modeling required must be do-able with the resources available, bearing in mind the need for training. We detail the best practice within Insiel in Section 4.1 and consider the technologies to be used. Where notations are new to Insiel, training is required which will impact on the resources available.
- The resultant model should demonstrate modeling of:
  - Structure. This is required to meet the objective of understanding the possible architectural structural reconfigurations, a priority for Objective 3.
  - Data and functionality. This is necessary so as to understand the impact of changes and to help to identify bottlenecks in the
delivery of the SoS functionality. These are priorities for Objectives 1 and 3.

- Concurrency. These are required to enable consideration of SoS-level properties we wish to analyse, such as safety, security and performance. These are priorities for Objective 2.
- Time.

- Given the structural model, the case study should provide an opportunity to perform change impact analysis as described in the case study objectives.
- Finally, the case study should provide a basis for setting out the expectations for the second CML-based case study, in Task T4.1.2.

## 3.6 Emergency Response SoS Outline

The ERSoS is composed of several constituent systems. The section will give a high-level outline of the constituent systems, their relationships and the functionality they provide to the SoS. The boundary presented here may differ for other scenarios and depending on the modelling taking place. In Section 3.6.1, we describe the boundary of the SoS, and in Section 3.6.2, we give a high-level overview of the functionality of the constituent systems.

### 3.6.1 Emergency Response SoS Boundary

In Figure 7 below, we present a high-level view of the SoS, its constituent systems (and the connections between them), the SoS boundary and the relevant entities in its environment. The constituent systems are: **Telecom, Phone System, Call Centre “CUS”, The Radio System, ERUs, Fire Brigade and Alpine Rescue.** The elements of the environment we consider in the case study constitute the **Caller, Target, Police and Hospital.**

![Figure 7 - Emergency response SoS boundary](image)
Figure 7 shows that the \textit{Phone System} and the \textit{Call Centre “CUS”} are together considered the \textit{Unified Emergency Service} shown in Figure 1. The \textit{Telecom System} is an important point of interaction with the environment, connecting to the \textit{Caller, Police} and \textit{Hospital}. Calls are routed though the Phone System to the \textit{Call Centre “CUS”} which manages emergency responses. Depending on the type of incident, a combination of ERUs, Fire Brigade or Alpine Rescue systems will be despatched to some target in the SoS’s environment by the CUS. The communication between these systems and the \textit{Call Centre “CUS”} is provided via a Radio System.

We consider the Telecom, Phone and Radio Systems as constituent systems. They express independent behaviours, each can be upgraded or substituted without affecting the overall SoS behaviours, and their performance is critical to the SoS functionality. The Police and the Hospital, however, are not designated as constituent systems for the purposes of our modelling here. This is because the SoS presented in Figure 7 is centred around the viewpoint of the operator of the co-ordinated emergency response operator, who employs the CUS as a management tool. Those services which are included as constituent systems are those which can be directly managed by the CUS (the Fire Brigade, the medical ERUs and the Alpine Rescue). This is not true for either of the Police or the Hospital, and so for the purpose sof our modelling we designate them as external to the SoS.

The SoS constituent systems are not designed to change unexpectedly during the delivery of services. However, problems may occur. Due to the mission-critical importance of the ERSoS, the SoS incorporates redundant “backup” systems for recovering operation during a fault, some of which are considered in the model. An example of this is a mobile phone system, which is used when faults occur in the \textit{Radio System}.

In the following sections, each of the constituent systems will be described in terms of their characterizing behaviours.

\textit{Telecom}

The \textit{Telecom} system provides connectivity between the \textit{Phone System} and entities in the environment such as the \textit{Caller, Police} and \textit{Hospitals}. The \textit{Telecom} may be operated by a range of service providers, from national landline operators to mobile phone operators. It may encompass routing exchanges, satellites, organisational systems, and a complex collection of hardware and cabling.

\textit{The Call Centre “CUS”}

The \textit{Call Centre “CUS”} performs three main tasks:

- \textbf{Manage emergency calls} – this encompasses the ability to receive, categorize, group and dispatch calls. The \textit{Call Centre CUS} interfaces with the \textit{Phone System} via human operators and software functions that include quick phone directory to forward the call to other organizations involved in the
emergency and tools to facilitate the emergency localization via telephone localisation services (GIS).

- **Identify the resources** – this functionality allows the operator to determine the appropriate response for an incident: in this task a human operator is supported by a Decision Support System which is part of the Call Centre CUS.

- **Manage emergency rescue units (ERUs)** – the Call Centre CUS interfaces with the Radio System via human operators and software functions. This task includes:
  - Schedule resources
  - Manage and monitor the transport
  - Health Care System alert and activation

Whilst currently considered out of scope, additional useful functions that could be added are GIS localization emergency data transmission to and from the ERU.

In the current implementation the human factor is still the most relevant. In fact, human operators are responsible for the communication with other constituent systems and external bodies involved in the emergency as well as managing the communication between the involved ERU and external parties (e.g. hospital).

**Phone system**

The Phone System is distinct from the Call Centre CUS as it is made up of third-party components that could be replaced with implementations from other vendors. It provides the possibility of handling calls on several lines, computer telephony and conversation recording. Furthermore, it includes a system for broadcasting emergency notifications (fax et cetera), and handles dedicated emergency lines.

In case of faults, some marginal components of the system (e.g. queue handling or conversation recording) can be bypassed, still maintaining the main functionalities, to ensure the provision of the public service.

**Radio system**

The Radio System is responsible for ensuring the communication with the ERUs. The system, which is an off-the-shelf-product supplied by third party, consists of a device and an antenna on board ERUs and connected to the Call Centre CUS that allows the communication to/from the Call Centre CUS. The Radio System also includes all hardware and software related to the relaying of radio messages. The local distributed devices decode and transform radio signals to audio conversation and/or to data messages to be properly handled by the Call Centre CUS.

The radio devices on board allow voice communication, as well as encoded communication via short messages for status update (started, on target, returning, ready for new mission).

In case of faults and to ensure communication of the ERU in those areas where the radio signal is shadowed (valleys, inside buildings), communication between ERUs and CUS is operated via the Mobile Phone system or the Telecom.
The ERUs, Fire Brigade, Alpine Rescue

For the scope of the Case Study, ERUs are ambulances and helicopters, equipped with a radio transmitter and receiver, capable of interfacing to the channels and protocols handled by the radio system, to ensure communication with the CUS and the Hospital.

3.6.2 Emergency Response SoS - Constituent System Functionality

The case study deals with health emergencies only. We do not consider public security, rescue of uninjured people or natural hazards that did not produce physical damage to people.

Emergencies are signalled by a phone call to the public health emergency number 118. The process of receiving and handling an emergency call can be summarised in the following steps:

- Queuing the call
- Answering and processing an emergency call
- Initiating the emergency response
- Servicing the emergency response

Queuing the call

A Caller initiates an emergency response by dialling 118 and connects to the Phone System of the SoS.

118 is associated with several phone lines therefore a 'busy-line' signal only occurs in exceptional conditions that we do not consider here. The Phone System places the call into an emergency calls queue. The queue is handled by the Phone System and displayed on the workstations for the first free CUS operator to manage it.

Answering and processing an emergency call

Answering the call triggers the following operations:

- The Phone System starts recording the conversation
- The call is removed from the incoming call queue and moved to the queue of calls in progress, so that all operators can see it has been answered
- Possibly, a form is presented to the operator, to be filled-in with casualty details obtained by interviewing the caller
- The incoming number field in the form can be filled automatically for fixed lines. Otherwise, this is filled manually by the operator
- Possibly, the operator views a map of the area of the reported incident

The caller, who may be different from the person/the people needing assistance, is then interviewed according to a protocol. Under certain conditions (hoaxes,
mis dialled number, absence of medical emergencies) the operator discards or diverts the call; the call data is marked with a discarded or diverted result flag. The conversation recording is automatically interrupted by the phone system as the line is closed and the call is removed from the queue of calls in progress.

Under normal conditions, the operator goes through all steps of the interview protocol. The interview gathers all elements to clarify:

- The mission priority (risk of life or not)
- The type and number vehicles needed (ambulance/helicopter)
- Which equipment should be available on the vehicle

**Initiating the emergency response**

Once it has been established that medical aid is required the CUS must identify a suitable set of ERUs. This is straightforward if there are sufficient idle ERUs available. If there are not the operator may decide to divert one or more ERUs already tasked on another mission, prioritising on the basis of the mission criticality as well as other factors such as the ERU’s current location and how close the ERU is to completing its current mission. If an ERU is diverted, the corresponding mission must be reassigned.

Finally, a message is sent to the selected ERU(s) communicating the target location and any other necessary details. The ERU(s) must send a confirmation and move towards the target location.

**Servicing the emergency response**

During the mission, the ERU sends status updates to the CUS (via Radio System) at specific milestones (e.g., reaching target location, leaving the target location, delivering the patient to hospital, available). The CUS operator inserts the status update messages in the incident form to make them visible to the other operators. Further details can be communicated during the mission from the CUS to the ERU in textual or vocal form via the radio system or mobile phone.

The mission is closed when the ERU involved in a mission returns to an idle state. If multiple missions are associated with a single emergency, the emergency response is closed when all missions are closed.

### 3.7 Emergency Response SoS Characteristics

Table 1 presents the eight SoS properties identified by COMPASS (see [Nielsen et al 2013]) and discusses the extent to which Insiel’s case study exhibits them. The eight properties provide a means for characterising the SoS and allow it to be compared more effectively to other SoSs.
Table 1 Characterising the Insiel case study using eight properties associated with SoSs

<table>
<thead>
<tr>
<th>Property</th>
<th>Insiel case study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autonomy</td>
<td>The constituent systems in the emergency response case study exhibit some autonomy. For example, one of the services that contributes towards the unified emergency services may make changes to the range of services they offer (e.g., fire brigade may extend their services to include rescuing persons trapped in damaged buildings or crashed cars). The CUS constituent system also exhibits some degree of autonomy; for example, a decision may be made to remove some constituent system from the SoS, or to add extra services and associated constituent systems to the current range (e.g., other emergency service teams such as the coastguard could be included).</td>
</tr>
<tr>
<td>Independence</td>
<td>Many (not all) constituent systems within this case study exhibit a considerable degree of independence. Each one of the constituent emergency services, for example, can and does sometimes operate as an independent unit without SoS involvement. The CUS constituent system does not exhibit the same level of independence. Without the participation of other constituent systems, including individual emergency services and communication networks, the CUS cannot deliver any meaningful functionality.</td>
</tr>
<tr>
<td>Distribution</td>
<td>Constituent systems are widely distributed over a geographical area. There are separate headquarters for the separate emergency services, and the CUS. Each constituent system itself experiences, in turn, a considerable degree of distribution, with individual mobile units covering a large geographical region.</td>
</tr>
</tbody>
</table>
| Evolution | The number of constituent systems and their interrelationships does have the potential to change over time in this case study. For example, a decision may be made to increase the services offered by the SoS by adding the coastguard, or another service not currently involved in the ERSoS. Relationships between constituent systems may alter over time; for example, it’s possible that constituent systems may develop new interdependencies between themselves if they begin to share a significant resource (such as a helicopter). The high level goals and requirements of the SoS also have the potential to evolve over time; for example, the central
government may introduce new requirements for response times.

| Emergence of behaviour | The emergent behaviour in this case study is found in the delivery of a unified emergency service, which can dispatch and monitor appropriate units from across multiple emergency services to respond to incidents. This functionality is not produced by the single constituent systems: the police cannot deliver emergency medical support alongside policing services if they operate in isolation - the participation of both the ambulance service and police is required. Without the participation of all constituent systems, the overall functionality of accepting a call for help on a single contact number and dispatching units that cross-cut different emergency services cannot be delivered. |
| Dynamicity of behaviour | Dynamically reconfigured behaviour is not strongly exhibited in this case study, although there are some limited opportunities for dynamic architectural reconfiguration. As a highly-visible, publicly-funded and safety-critical system, the SoS must deliver its intended functions with high degrees of reliability, transparency and traceability. Therefore unexpected dynamic reconfiguration of relationships between constituent systems, or removal of a constituent, is not regarded as desirable, as it is likely to result in confusion between human participants on the ground, a degraded level of emergency response service and gaps in the records. Where necessary, acceptable architectural reconfigurations are identified and agreed in advance. Within these constraints, however, the system is capable of demonstrating some dynamic reconfiguration. For example:
  * If the radio signal is insufficient an alternative communication channel is employed (e.g., mobile phones). If the planned backups are not available this may mean adopting unplanned alternatives dynamically (e.g., using a patient’s phone, finding a pay phone).
  * It is conceivable that one emergency service may be unable or unwilling to participate (e.g., a strike action may temporarily make one emergency service unavailable). In this case the SoS can reconfigure by locating alternative cover from another source. For example, if ambulance drivers are on strike, the military may be required to help provide cover. |
| Interdependence | The interdependence of constituent systems in this case |
In general, the CUS system, which interacts with most of the constituent systems, may need to accept altering some of its functionality if a decision made by an emergency service provider demands this. However, other constituent systems are not able to accept significant possibility of sacrificing some functionality in order to participate in the SoS.

### Interoperability

The case study does exhibit interoperability challenges. The constituent systems of the SoS include elements of software (such as the CUS), hardware, third-party systems (such as a communications network, GPS maps and devices) and a large number of diverse human participants. The separate constituent systems exchange information between themselves, or sometimes with the environment, in widely varying formats – e.g., medical records or GPS co-ordinates. In addition, there is a large amount of both structured and unstructured data exchanged between the human participants of various constituent systems which must be translated into formats suitable for exchange, storage and use (e.g., the CUS operator will extract details of location and classification of the incident from the original caller's first report).

In summary, we believe that the emergency services case study can be characterised against the properties as follows:

- **Autonomy**: constituent systems exhibit a high level of autonomy
- **Independence**: most constituent systems exhibit high independence; the CUS exhibits no independence
- **Distribution**: the SoS is highly distributed
- **Evolution**: the SoS potentially evolves over time, but does not exhibit strong examples currently
- **Emergence**: the SoS exhibits a high degree of emergent behaviour
- **Dynamicity**: the SoS currently exhibits only limited dynamic reconfiguration
- **Interdependence**: the CUS constituent system exhibits high interdependence; other constituent systems exhibit low interdependence
- **Interoperability**: the SoS exhibits a high degree of interoperability
4 Baseline Technologies

4.1 Commonly adopted technologies in Insiel

Since its beginning in 1974, Insiel has developed ICT products and systems for Public Administrations in Italy. Insiel is fully aware of the importance of adopting proper methods and tools, both to support development of high quality products from early stages in the software life cycle, and to support maintenance of a portfolio of assets in a technological and methodological environment that is constantly evolving.

In recent years UML has become the most widely-used modelling languages at Insiel. Different tools are employed, depending on the domain problem and operational unit, with varying vendor specific or open source implementations like IBM Rational suite\(^1\), StarUML\(^2\) or ArgoUML\(^3\). Although Insiel personnel are familiar with UML notations, tools and methodologies, when we approached the ERS/CUS Case study modelling we faced the following challenges:

- The available analysis/design documents were written using a non-UML-based methodology which preferred the use of free text to any formal methods
- All documents were written in Italian, which made them difficult to be shared within the international consortium
- The methods employed were not SoS-specific

All these factors combined make it difficult for COMPASS project members to understand clearly the interfaces among the various systems. In order to cope with the project objectives it was necessary to have a consistent model. Therefore the first challenge was to select the proper technologies and methods to be used in task T4.1.1. Technologies previously used were not considered appropriate for the following reasons:

- They are not specifically SoS oriented
- Insiel has a strong interest in widening its experience in the domain of SoS modelling as its current evolution-trend is towards system integrator
- ICT are becoming more and more complex moving towards the SoS - when combining different types of “things” (see the Internet of Things paradigm\(^4\)) the pure software approach is not sufficient.

---


\(^3\) [http://argouml.tigris.org](http://argouml.tigris.org)

4.2 Selected technologies for Task 4.1.1

The available baseline technologies were initially presented during the first convergence meeting in Newcastle in October 2011. SysML and VDM were found to be suitable baseline technologies for Task T4.1.1 modelling objectives. SysML requires a short learning curve for our UML-skilled resources and SysML models provide a good overview of the problem domain. VDM uses a formal representation that, although more difficult to learn, seems the adequate tool to formally drill-down into specific issues, thus providing a more detailed description of interactions than SysML, together with simulation capabilities and proof obligations.

4.2.1 SysML

The Unified Modelling Language (UML) [2] is an established notation for modelling software systems. UML has recently been extended and specialised for systems engineering to create the Systems Modelling Language (SysML) [3]. The two notations have a lot of overlap, but where UML focuses on modelling the structure and behaviour of software systems, SysML uses the more general Block to represent a broader notion of systems, including SoS.

SysML is a graphical language with semantics given in “precise natural language”. It provides nine different diagram types to represent systems requirements, structure and behaviour. Different aspects of the system are modelled in each diagram type and the combination of all the diagrams constitutes a model of the overall system. It enjoys wide industrial support and a sound tool base, including the Artisan Studio1 suite of tools, which provides support for creating SysML models and checking their consistency.

Besides its similarities with UML, SysML has other characteristics that made it interesting for Insiel, specifically for the ERSoS case study. First of all the possibility to include in the model also other types of systems allows Insiel to explore and understand the influence of external factors that are, in most of the cases, responsible for delays and unfulfilled service. In a pure UML model such factors are impossible to model and their influence on the model needs to be simulated through different types of algorithms. For instance the event of an ambulance that breaks down could be modelled imposing an artificial lag in the time of arrival, but in this way we could only approximate the impact of such an event and we can not investigate all the implications of such an event on the SoS.

4.2.2 VDM

The Vienna Development Method (VDM) [4] provides a set of model-based specification languages and a proof theory to enable key aspects of a system to be modelled and analysed at varying levels of abstraction. In this deliverable the

1 Artisan Studio: http://www.atego.com/products/artisan-studio/
object-oriented dialect VDM++ [5] is used. VDM has industry-strength\textsuperscript{1} and open-source\textsuperscript{2} tool support.

A VDM++ model is strongly typed and describes the behaviour of a system through operations and functions that are organised in classes. Operations can either describe behaviour explicitly through statement blocks or implicitly through pre- and post-condition pairs. Important properties over types and the state space can be asserted using invariants. Object-oriented features such as inheritance are also supported. Lower level features such as threads can also be specified, but these are not used in the models presented in this case study.

Overture, which shall be used in this case study, provides an IDE with syntax and type checking as well as simulation and advanced testing features like formal proof and combinatorial testing (this feature enables high volumes of test to be generated over a specified range of inputs), thus extending modelling to a more comprehensive and accurate approach.

As explained before, we foresaw great potential and benefits in the use of VDM that go beyond the detailed description of interactions, simulation capabilities and proof obligations described before. Although less intuitive, its formal representation appears to be more agile and therefore useful to quickly model fine details of a given issue. The strong similarities with Object Oriented languages like Java make VDM++ particularly attractive for those who prefer a bottom-up approach.

As we expect the tools we are currently exploring will be enriched with the COMPASS modelling language extensions in the coming phases of the project it is expected that these improvements and the use of CML in the accident response case will be described in D41.2 accident response engineering analysis using COMPASS methods and tools.

\textsuperscript{1} VDMTools: http://www.vdmtools.jp/en
\textsuperscript{2} Overture: http://www.overturetool.org/
5 Case Study – Accident Response System

The Case study has been described using the two selected languages, as specified in Section 4. SysML was used to create global models of the system spanning the various independent systems detailing interaction points, depicting entity relationships in disparate models and exhibiting data models across different systems databases based on processes executed. This methodology was useful to have an overall graphical representation of the system and to outline the principal behaviours, while VDM was used to understand the behaviour and check models under a series of different conditions of specific processes.

In the remainder of this section, both approaches are described – SysML in Section 5.1 and VDM in Section 5.6, with the complete models are provided in Annex 1 and Annex 2 respectively.

5.1 Approach to Benchmarking

We will use the following benchmarks to assess the effectiveness of SysML and VDM each for meeting the SoS modelling requirements of Insiel. The benchmarking approach adopted by COMPASS is explained in full in D11.2. Each of the following benchmarks addresses a need that Insiel has; SysML and VDM will be awarded a value for each between 0-4, based on Insiel’s experiences with the tool. In Phase IV of the project the same benchmarks will be used to assess the performance of the COMPASS tools in a similar task.

- **SoS Model Identification (SMIA)**
  The model-based method and SysML must provide aggregation and modelling capabilities allowing identification and incorporation of SoS elements and architectural patterns

- **SoS Model Development (SMDA)**
  The model-based test method must provide guidelines, architectural patterns and modelling capabilities for development of SoS test models.

- **SoS Architectural Modelling (SDAA)**
  The modelling language must provide Systems of Systems modelling capabilities for developing optimal architecture

- **SoS State Transition Relation (STRA)**
  The model-based method must support modelling of states and states transitions for both the constituent systems level and the SoS level. The transition systems of the test models must be verifiable

- **SoS Model Verification (SMVA)**
  The model-based method and modelling language must provide guidelines and modelling capabilities for simulation of use-cases, and sound SoS reasoning. Simulation results must be decidable

- **SoS Change Propagation Analysis (SCPA)**
  The model-based method should provide excellent support for identifying dependencies between constituent systems and the likely candidates that will be affected if changes are introduced by a constituent or the environment
- **SoS Interfaces Analysis (SIAA)**
  The model-based method should provide excellent support and guidelines for the identification and analysis of interfaces and contracts both internally and externally.

- **SoS Emergent Behaviours Requirement (EBRA)**
  The model-based method must support development of emergent behaviours requirement that conform to standard requirements quality attributes. An EBRA must be consistent, testable and traceable.

- **SoS Fault Tolerance Analysis (SFTA)**
  The model-based method should provide excellent support and guidance of the analysis of fault tolerance, by helping to identify possible faults, events that trigger them and supporting the development of recovery strategies.

- **SoS Test Model Development (STMA)**
  The model-based test method must provide guidelines, architectural patterns and modelling capabilities for development of SoS test models.

The key objectives for Insiel have already been introduced (in Section 3.3). A key problem that Insiel would like to tackle concerns change impact, and mapping proposed changes to a costing model. A change impact model relies heavily on being able to identify dependencies between constituent systems, or between constituent systems and their environment. If this information is available it can be used to identify those constituent systems which may be affected by a change. This motivates the inclusion of the benchmark SCPA, as well as the analysis and identification of interfaces that underpins it (assessed by SIAA). Insiel operate in a domain with a high requirement for fault tolerance and so support for planning fault tolerance across an SoS is also a feature that Insiel wish to assess.

Objectives also pose the question of analysing SoS architecture, and identifying an optimal SoS architecture given a selection of constituent systems and desired emergent behaviour. Benchmarks included here some key functions which are necessary in order to reach the point where different architectures can be analysed and optimised; this includes being able to develop architectural models easily; identifying architectural features and boundaries; identifying interfaces; and managing emergent behaviour.

### 5.2 Approach to SysML modelling

Given the Emergency Response SoS case study, as introduced in above we may begin modelling the SoS using the selected baseline technologies. Due to Insiel’s previous experience with UML and RUP¹ (Rational Unified Process), the SysML model was the first to be defined. This approach is composed of the following steps that have been derived and adapted from the RUP:

---
• Define the **boundary** of the SoS. This defines the separation between the SoS detailed in the case study and the rest of the world, it is useful to identify the flows of information that enter and exit the SoS.

• List the **requirements**. These provide information of the topic to be analysed, are usually divided in:
  - **Functional Requirements** - representing actions and behaviours exposed by the SoS generally described by use cases
  - **Non Functional Requirements** - representing all other requirements that are not functional, therefore related to other constraints including legal constraints, standards and specific hardware to be used.

• Produce the **overall SoS Use Case Model (SoS-UCM)**. Like the Business Use Case Model in RUP, this is a high level use case diagram illustrating the scope of the SoS being modelled. The diagram identifies the **boundary** and also identifies business **actors** (roles played by organizations, people, systems external to the business) and the **functions** (behaviours) they request from the SoS. The SoS-UCM should include all macro-functions (high level) that the SoS will perform and should contain a limited number of use cases (5-6).

• Produce the structural and behavioural diagrams corresponding to the SoS constituent systems and the behaviours identified in the SoS-UCM.

Also for the concept of Actor and Use Case we refer to UML: **Actor** is anything that interacts with the SoS, it could be an IT system, a role within an organization (e.g. CUS operator). The same person using the system may be represented as two different actors because they are playing different roles. A **Use Case** defines the interactions between external actors and the system under consideration to accomplish a goal. We overview the steps of the SysML approach we followed to model the SoS in the next sections.

### 5.2.1 SoS boundary

The SoS boundary of the SysML model is defined as it appears in Figure 7. Section 3.6.1 details the minimum set of constituent systems sufficient to model an “emergency rescue mission” and tackles the issues we were interested in resolving from an SoS perspective. However, we do not include the TeleCom constituent system from Figure 7 in the SysML model. The model largely focuses on the CUS constituent system and the connections between it and the phone system and the ERU via the radio system.

We use the SysML structural block definition diagrams to describe the SoS boundary and to show the constituent systems of the SoS considered in this model. The SysML block definition in Figure 8 shows that the SoS is considered to include the **Phone System**, **Call Centre “CUS”**, **Radio System** and several **Emergency Response Units (ERUs)** including ambulances and “flycars” (swift response vehicles).
The SoS constituent systems are not expected to change in unanticipated ways during the delivery of services. However, fault conditions may occur; some of which are considered in the model. The Phone System and Radio System, from a behavioural point of view, could be considered simple interfaces respectively between the public Telecom system and the CUS and between the ERUs and the CUS. However they are considered to be constituent systems because: they are made up of third party components; they are relevant for fulfilment of non-functional requirements such as performance; and they are critical for possible failure conditions. The Phone System and Radio System may be independent. However, in some circumstances some resources can be shared such as conversation recording devices.

The second block definition diagram, in Figure 9, depicts an SoS, where redundancy is increased by adding an additional constituent system to provide “backup” communications. In this case, the Mobile Phone System is connected to the ERUs, Phone System, and Call Centre “CUS”. The addition of this system provides fault tolerance for communication between the Call Centre “CUS” and ERUs in case of the failure of the Radio System. This is a proof of concept of modelling the addition of a new system to the SoS.
5.2.2 Requirement Classification and Catalogue

The functional requirements for the SoS are detailed in the SysML requirements diagram in Figure 10. The diagram identifies the various requirements for the SoS and their relationships. Each requirement is given a unique name or the form "req_ Category_XXXX" where 'Category' refers to the part of the SoS the requirement relates to (for example req_CUS_000000 is a requirement related to the Call Centre “CUS” constituent system, the other categories are: Auth, referring to authorization issues; Radio, referring to Radio system; and Phone, referring to Phone system). Each requirement identified in the diagram also contains a textual description and, where relevant, identifies which SysML model element satisfies the requirement (for example the req_CUS_0101 requirement is satisfied by the New call info operation. A similar process was carried out for non-functional properties, as found in Annex 2.

Given the requirements identified in Figure 10, we tabulate the requirements and output them in table form – an extract of which is shown in Table 2.
<table>
<thead>
<tr>
<th>Name</th>
<th>Req</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_Auth01</td>
<td>DEFINE OPERATOR ROLES The System shall allow</td>
<td>configuration of operations allowed to each operator or role of an operator. An operator can change role.</td>
</tr>
<tr>
<td></td>
<td>DEFINITION OF USER PROFILES/ASSIGN ROLES TO</td>
<td>USERS When user is authenticated, the System shall enable operations on the basis of the roles the user can embody.</td>
</tr>
<tr>
<td>req_Auth02</td>
<td>TRACE OPERATOR/ROLE ACTIVITY The System shall</td>
<td>trace operations performed by operators, with particular attention to accessing private data of patients and other people.</td>
</tr>
<tr>
<td></td>
<td>RETRIEVE SETTINGS AND WORKING PREFERENCES OF</td>
<td>OPERATOR/ROLE When a user is authenticated in the system, if the user is a CUS operator, the workstation user interface will be set up with personal preferences including frequent calls directory and headset volume.</td>
</tr>
<tr>
<td>req_Auth03</td>
<td>ACTIVATE EMERGENCY RESPONSE FOR EACH INCOMING</td>
<td>CALL, (general requirement, detailed by sub-requirements) The CUS system (with intervention of operator) assesses response needs and activates required resources.</td>
</tr>
<tr>
<td>req_Auth04</td>
<td>MAPPING FACILITY This includes browsing maps and</td>
<td>other features that do not need detailing at the SoS level.</td>
</tr>
<tr>
<td></td>
<td>MANAGE AND SAVE CALL INFORMATION (general</td>
<td>requirement, detailed by sub-requirements) For each call processed, a record shall be saved containing the number of caller, essential information (form with casualty details if needed) and result of operation. Not pertinent calls (misdialed number and so on) and calls diverted to other organizations shall be logged as well with a proper result code.</td>
</tr>
<tr>
<td>req_CUS_000000</td>
<td>CREATE 'RECORD' FOR CALL / SAVE IT WITH TYPE</td>
<td>OF ACTION FLAG WHEN COMPLETED. see &quot;Call info bag&quot; in model</td>
</tr>
<tr>
<td></td>
<td>PROVIDE USER INTERFACE FOR RECORDING INFO</td>
<td>PROVIDED BY THE CALLER The CUS software shall enable</td>
</tr>
</tbody>
</table>
The complete set of requirements is listed in Annex 2.

5.2.3 **Produce the overall SoS Use Case Model (SoS-UCM),**

Having defined the SoS boundary, the resulting SoS-UCM is represented Figure 11 Overall SoS Use Case Model.

---

**Table 2 Example of functional requirements collected**

<table>
<thead>
<tr>
<th>Name</th>
<th>Req</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>req_CUS_0103</td>
<td>ASSOCIATE CALL WITH RESPONSE EVENTS IT GIVES RISE TO. “Call info bag” logged after a call should be an entry point to access response and mission data</td>
</tr>
<tr>
<td></td>
<td>req_CUS_0104</td>
<td>DISPLAY RECENT CALLS INFORMATION Incoming emergency calls shall be displayed in the user interface of CUS software with number and other relevant information. This is a different requirement from &quot;req_Phone0101&quot; as here the focus is on the incoming calls and having them attached with info.</td>
</tr>
</tbody>
</table>

Figure 11 Overall SoS Use Case Model
The main Use Cases identified and reported in the diagram consisted in those related to the emergency rescue mission are:

- Receive an emergency call
- Alert health structure
- Schedule response mission
- Monitor rescue and resources

Although the competent health structure is alerted just before sending the rescue unit to the target, this use case has been placed outside the SoS boundaries.

### 5.2.4 SoS Internal Structure and Behaviour

In this section, we consider the structural and behavioural diagrams describing the SoS SysML model.

**SoS Internal Structure**

Given the high-level composition block definition diagrams identified in Section 5.2.1, we may consider the internal connections and interfaces between the constituent systems. Given the scale of the SoS and the large number of connections for the different functionalities of the SoS, we provide several internal block diagrams.

Figure 12 shows the interfaces between the *Phone System, Call Centre “CUS”* and *Radio System* constituent systems of the SoS. Interfaces are attached to ports on the boundary of the constituent systems to denote points of interaction between the systems. This diagram shows the external interfaces between these constituent systems as a high-level overview.
Figure 12 - Constituent system external interfaces

Figure 13 below details the interfaces and connections corresponding to the scheduling functionality of the Call Centre “CUS”. In this diagram, the internal components of the CUS constituent system, relevant to the scheduling of ERUs, are identified. A subset of the external ports and interfaces identified in Figure 12 are shown, with connections made to the internal components providing and requiring the interfaces of those ports. The figure identifies the Map Provider, Webserver, CUS Client and Human Operators as those internal parts of the Call Centre “CUS” that are involved in the ERU scheduling, communicating with the ERUs via the Radio System and Mobile Phone System.
Figure 13 - Internal Block Diagram depicting CUS components and connectivity for ERU scheduling

Figure 14 below details the interfaces and connections corresponding to the Phone Answering functionality of the Call Centre “CUS”. The diagram is similar to Figure 13 in that it presents a subset of the internal structure of the Call Centre “CUS”, in this instance with respect to the functionality of phone answering. In this diagram, the internal CUS components of interest include: Webserver, Map Provider, CUS Client, Emergency Phone Appliance, Human Operator and CUS assess severity and required resources. Notice that some of the internal components are involved in multiple Call Centre “CUS” functionalities – which shall be described in more detail in the SysML behavioural diagrams.
The final internal block diagram corresponding to the functionality of the Call Centre “CUS” is given in Figure 15, detailing the interfaces and connections corresponding to the ERU monitoring functionality. Externally, the diagram is identical to the interface connectivity in Figure 13 however. The Call Centre “CUS” internal structure differs due to the use of different internal interfaces between the Call Centre “CUS” components.
The internal block diagram in Figure 16 shows the internal structure of the Phone System and Call Centre “CUS” that are involved in the phone handling and call recording functionality of the SoS. Whilst the Phone System is managed and owned by an external organisation, the internal composition is known and its components are shown. Notice, however that the diagram shows the connections between the Phone System components, but not the interfaces. It is clear that considering the internal structure of the constituent systems of the SoS increases the complexity of the diagrams and resulting model.
Figure 16 - Internal Block Diagram depicting CUS and Phone System components for call handling functionality

The interfaces identified in Figure 12 are described in more detail in Figure 17. The interfaces in are given as SysML interface blocks, with the interfaces defined in terms of the operations provided, and the operation signatures. For example, IIF conversation handling interface provided by the Phone System and required by the Call Centre “CUS”, consists of two operations: start recording and stop recording. The diagram also begins to identify relationships between interfaces, with the IFF CT Phone queue handling, IIF CT Phone conversation handling and IIF CT Phone queue info interfaces composed into the IFF CT Phone handling interface.
It should be noted that, whilst the Internal Block Diagrams for the Radio System and ERUs will be complex, their internal representations are not available for modelling – they may only be represented by the collection of external provided and required interfaces.

**SoS Behavioural Modelling**

The SoS-UCM in Figure 11 Overall SoS Use Case Model, along with the block definition diagram in Figure 18 showing the behavioural processes, we began considering the behaviour exhibited by the emergency response SoS. The processes identified in Figure 18 each have an activity diagram associated with it to describe aspects of the behaviour. Sequence diagrams are also defined to demonstrate scenarios. In this section, we present one of these activity diagrams (and the related sub-diagrams) and also a selection of sequence diagrams. The remaining activity diagrams we developed may be found in Annex 2.
The block definition diagram in Figure 18 identifies several processes including *Initiate Response, Process Call, Replay of recorded conversation* and *Time Mission*. The processes describe the different actions that may be performed during the life of the SoS, here. In this section, we consider the activity diagram corresponding to the Process Call process, as shown in Figure 19.
Figure 19 - Activity diagram for the "Process Call" behavioural process

Figure 19 shows that there are two constituent systems involved at this level of this activity: the Phone System and Call Centre “CUS”. The process begins with the...
Phone System receiving an Emergency call signal, which results in the Phone System performing the Enqueue call and Record conversation activities. The activity diagram details the subsequent flow of activity resulting from the incoming emergency call. There are a few elements of interest in this diagram which we detail here.

First, there are a number of activities on the figure which are themselves placeholders for sub-activity diagrams: Answer call, Obtain casualty details, Replay of recorded conversation, Integrate response event detail, Obtain further detail and Save call result. These diagrams are given in Annex 2, though we provide the activity diagram for Obtain casualty details activity below in Figure 20 to demonstrate the flow of control.

The second point of interest is the flow of activity subsequent to the Obtain casualty details activity. Once the CUS operator has ascertained the details of the casualty, they have several choices on how to proceed. This is demonstrated using the diamond decision node syntax. The choices include a failed call, discarding the call (due to a prank call), creating a new event, adding more detail to an existing event or diverting the call to an external organisation (such as the police). This process may start other processes (Start response and Update response) and ends when the Phone System detects an end of conversation and stops recording the conversation.
Having defined the various processes, the SoS behaviour was further modelled using a collection of SysML sequence diagrams. These diagrams describe the flow of information among constituent systems in given scenarios. We illustrate this using the diagram below in Figure 21, as has been mentioned earlier – the remainder of the model is given in Annex 2.
This sequence diagram describes the flow of information between the Phone System and CUS constituent systems. In this scenario, an emergency call is received from the environment (as the signal mentioned in Figure 21), which starts the Phone System activity of queuing the call. When the CUS answers the call, the Phone System starts recording the call in parallel with the CUS obtaining details and processing the call. In this scenario, the call is a new event. Once the call is handled, the Phone System ends the recording. Also noted is the response time, from the time the call is made, through to the time the call result is saved.

5.3 Challenge problem analysis in SysML

This section returns to the challenge problems that were introduced in Section 3.4, and considers how each challenge could be approached based on the SysML models that have been developed already. Tackling the challenge problems allows us to address some of Insel’s major objectives for the project. We address only a subset of the four challenge problems here; a complete analysis of all four challenges remains for future work.

5.3.1 Evolution of a constituent system

In Section 3.4 Challenge B considered the problems that arise when a constituent system implements some change. As an example, consider the relation between the CUS, Radio System and ERU and the process of ERU status and position monitoring. Currently this process relies on the human operator listening to radio vocal communications with ERUs and manually updating the status of each ERU accordingly. This approach is not very precise and has a high probability of introducing further errors. However, localization technologies are mature and we consider upgrading the ERUs so that they can directly communicate status and position to the CUS. This requires a change to the radio system, adding the capability to filter data from voice and dispatch it via a dedicated interface. Such an upgrade might imply changing data radio transmission protocol (vendor specific and/or encrypted) and also a change to the device data transmission interface. Moreover, it requires ERUs to carry on-board GPS and the
implementation of communication channels and protocol between ERUs and the CUS through different usage of the Radio System.

The current functionality is modelled in SysML in Annex 2, both in terms of a series of processes, and (as a sequence diagram) as a sequenced series of exchanges. Creating the sequence diagrams has been helpful for developing more precise interfaces for information exchange, although because this particular example models human speech activities the information exchanged is not formally structured.

- Figure 53 in Annex 2 shows from point of view of the CUS operator requesting location information from the ERU.
- Figure 55 depicts the general process of radio communications between the ERU and the CUS operator. Further details of specific exchanges that are regularly required are modelled in subsequent diagrams.
- Figure 55 depicts the general process of the CUS operator radioing the ERU, and presents the process of the CUS operator monitoring the ERU’s radio messages.
- Finally, Figure 63-Figure 67 depict the behavioural aspects of the exchange of information between the ERU and the CUS operator, as a series of sequence diagrams. Figure 67 in particular shows a modified version of the sequence diagram that allows for automatic transmission of location information instead of human communication.

The original specification of Challenge D (add a new constituent or modify an existing one so that architecture is changed) posed the question: how does the proposed change impact on existing SoS properties, such as fault tolerance? We have presented some techniques for exploring this further in SysML; Figure 67, for example, depicts the “correct” flow that we normally expect to see and also some alternative sequences that can be adopted instead if the radio is unavailable.

This particular modelling technique is quite intuitive and helpful for identifying the sequence of events, the minimum series of interactions that are necessary and the sequence as we move from one sequence of events to the recovery sequence. However, it’s possible for sequence diagrams which are depicting “normal” and “recovery” behaviour to become very complex quite quickly. It’s not always clear what events are triggering the shift from one sequence to another, and it’s also not always clear what state we have finished in after the recovery sequence is carried out. Therefore effects of the recovery sequence on neighbouring sequence diagrams may be difficult to detect.

5.3.2 A new environment

In Section 3.4 Challenge C introduced the challenge of installing a new version of the SoS in a brand new environment, which may be subtly different to the original environment in ways which have not been anticipated. We have not created SysML models of this particular scenario, but here we discuss how our existing SysML models can be leveraged to answer some of the questions
associated with this challenge, and which of our models would need to be revisited.

We can assume that there will be different environmental challenges or requirements in the new location. For example, perhaps the new region is more remote and/or mountainous than the current regions, with an increased risk of ERUs attending incidents without radio or mobile coverage; or it may be a coastal region, with a requirement to incorporate the coast guard in the ERSoS; or there may be different protocols for communicating with regional fire brigade or police force.

Firstly we require a precise specification of the external interfaces and the environment in the current location, for comparison with the new location. We have presented our initial external interfaces model as Figure 12. For example, the radio and phone interfaces for communicating internally, with ERUs, and externally, with other emergency services, would need to be compared between the two locations. This is not a simple task; it requires careful detailing of the fine-grained sequences and data exchanges between human actors who communicating partly with automated radio signals and partly with freeform human speech. Although Figure 12 helps us to identify where interfaces between two or more parties occur, and the data that may be exchanged between them, we need to know if there are other dependencies or changes. It's possible that the same data is exchanged but in a different sequence, and this may violate the assumptions made by the existing ERSoS. For this, we need to compare carefully the sequence models and process models for the two locations as well as the interfaces. Under Challenges B and D we discuss further techniques we believe can be employed for analysing whether interfaces have changed.

We present a selection of use cases in Section 12.1. Use case diagrams are the first point for modelling the expectations of the users in the new environment and so the use cases should be revisited. This would be an additional means of eliciting from the key actors in the new environment their expectations and requirements, which may differ from the current environment. However, linking use cases to the interface and sequence diagrams is not necessarily straightforward In SysML, because the two models represent the system in very different ways.

One aspect that is missing from SysML is the ability to make design decisions based on SoS performance. For example, Insiel has an interest in identifying where bottlenecks and delays occur when the SoS is responding to multiple emergencies. If a delay occurs, Insiel would like to be able to analyse whether it is because ERUs are inadequately distributed, because the shifts have not been properly optimised or some other reason. We discuss our thoughts on a VDM model designed for load analysis of the system at runtime in Section 5.7.1.

5.3.3 A new constituent system which changes the architecture

In Section 3.4 Challenge D introduced the challenge of upgrading one of the constituent systems, or of introducing a new system in the SoS, such that the SoS
architecture is altered. This requires a complete understanding of the existing interfaces among the constituent systems; for instance, we may need to change one of the constituent systems due to changes in the constituent system contracts, or to introduce a new constituent system to take advantage of new technologies. Being able to identify changes propagated across the system like this is an important factor for Insiel; this has very practical uses as the foundation for a change impact estimate.

As an example case, we considered the introduction of a mobile phone system in the ERS to support the communication between the ERU and the CUS, especially in areas with a low radio coverage. This may potentially alter the system architecture, as there will be a new dependency introduced between the existing radio system and the mobile system; one constituent system will only be activated depending on the behaviour of the other.

Modern mobile phone technology is mature and a single interface seems to be sufficient to include voice communication, messaging and data exchange. The major challenge in this case is that the operator on the ERU will have another device to deal with. At times it may not be clear to the operator which method an ERU is using to communicate. This particular proposed change has been modelled in SysML diagrams in Appendix 2.

- Figure 39 and Figure 40 some higher-level processes of interaction between the CUS operator and the ERU (CUS scheduling and CUS monitoring respectively) with the mobile phone system added as a new constituent system. We can see that ideally Figure 55 (presenting the process of a general radio communication) could be updated with the new constituent system as well; this would enable us to present a more generic process for identifying a radio error and switching to the backup mobile phone, and this process could be inherited as appropriate by detailed models derived from this one.

- Figure 55 (depicting the process of the CUS operator updating the CUS record with location radioed by ERU) has also been updated with the new constituent system. This figure is an adaptation of Figure 63, which presents a similar “baseline” process without the backup mobile phone constituent system.

- Figure 66 and Figure 67 show updated versions of the CUS operator updating the CUS with ERU location information, based on data obtained by radio. For comparison, Figure 65 presents a similar, unaltered “baseline” process without the mobile phone system.

Although it has not been updated, it’s clear that Figure 37 will need to be revisited now that we have added the mobile phone system; this figure presents the external interfaces between the Phone System (used by members of the public to call emergency services), the CUS and the Radio System used by ERUs. Use cases would be a good way to identify, for example, if extra emergency services such as the coastguard must be incorporated. In addition, we should also consider that Figure 12 would also need to be updated with the new
interfaces that now exist between the constituent systems (including the addition) and with the environment.

As with other challenges, SysML provides an intuitive means for identifying and comprehending the processes, the sequences and the interfaces that exist between constituent systems. SysML does help to identify some constituent systems in the SoS that may need to make changes based on the addition of a new constituent.

5.4 SysML Modelling Evaluation

This section presents some observations based on Insiel’s experiences of creating SysML models of the ERSoS.

SysML language evaluation

The SysML language and tool support such as Artisan Studio allow a very detailed description of the SoS, although this is true only if all required information is inserted into the tool. The initial difficulty we experienced was in determining the correct process of constructing a complete model for the purposes of achieving the objectives set for this task. As the ERSoS is an existing SoS, we attempted a top-down analysis from an abstract description to a more detailed version, which had to be matched with the existing descriptions of the constituent systems – typically recorded in different notations. Understanding which diagram types are required, the ways in which they may be used and how to use the range of diagrams to complement each other to produce a consistent model was a considerable challenge and required a lot of effort.

Whilst the diagrammatical representation afforded by SysML can be very communicative and intuitive, each diagram has background information to explain the relationships between the individual diagram elements and diagrams. This takes advantage of the Artisan Studio hyperlink functionality to aid navigation through the model. This text is reproduced along with each diagram in Annex 2.

Once the SysML model was complete, the task of ensuring the diagrams were readable was a considerable task. The initial diagrams produced large views that were difficult to exploit in printed documents. As such, a series of iterations were required to ensure the diagrams described smaller subsets of the models – making the diagrams more focussed. On reflection, this had the benefit of producing diagrams that were easier to understand, such that the background descriptive text included for each diagram made clear the purpose and relations to other diagrams.

With respect to the Artisan Studio toolset used for SysML modeling, our main problems were related to the time taken to learn how to construct models, given our previous experience using the UML Rational tools, as described in Section 4.1.
**Case Study Model Evaluation**

The use of textual descriptions is necessary to clarify the meaning of similar diagrams representing ‘baseline’ and ‘improved’ behaviors or constituent systems. In fact, there is no tool to represent evolution of the SoS within a same model.

The challenge of properly representing concurrent access to resources by independent constituent systems and processes does not appear clearly in the models. The activity diagrams include some decision nodes regarding this, however, further effort is required in future work.

The representation of failures and failure recovery is an initial attempt. Using a signal to raise a failure event is a quick solution for introducing failure into an existing model, whilst a detected frequent failure might be modeled with decision nodes (for example, if the radio does not connect, try a mobile phone). We propose to work closely with Task 2.4.2 in subsequent work to represent faults and reason about SoS failure.

### 5.5 Benchmarking SysML modelling

This section presents benchmarked evaluations of the SysML modelling techniques. The benchmarks are allocated a value from 0-4. The benchmarking process is explained in detail in D11.2.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SoS Model Identification (SMIA)</strong></td>
<td>2</td>
</tr>
<tr>
<td>We find that SysML does provide generally good support for identification of SoS elements, even in a case study like the ERSoS where the boundaries of the system are not clearly-defined. However, there is little support for architectural decision-making, even if designing architectures is supported.</td>
<td></td>
</tr>
</tbody>
</table>

| **SoS Model Development (SMDA)**      | 2     |
| We find that SysML’s support for developing SoS models to be good. However, we are still lacking some architectural guidance, as mentioned above. The ERSoS is quite complex for political and organisational reasons and the boundary of the SoS is particularly difficult to define. For the end user who calls for emergency assistance, for example, the ERSoS boundary naturally seems to include bodies like the police and the local hospital emergency rooms. However, from the perspective of the CUS operator, the police and hospital may be considered extra constituent systems (albeit operating with less interaction with the CUS than other constituents) or they may be considered external bodies (albeit interacting with the CUS rather more than other agents within the same environment). There is no clear guidance on the best course of action, although generating some models demonstrating the interfaces between different organisations and systems can help to show possible locations where the boundary can be drawn. |       |
### SoS Architectural Modelling (SDAA)
SysML is naturally very amenable to modelling architectures, rendering different system architectural styles graphically for easy comprehension, and linking these to processes on activity models and sequence diagrams for more fine-grained modelling of behaviour. Although different architectural models can be generated using SysML, however, there is very little guidance on how to select between architectures for an SoS. It’s not straightforward to identify from an architectural SysML model what types of trade-offs may be made by selecting one architecture above another.

| 1 |

### SoS State Transition Relation (STRA)
SysML does encompass techniques for state transition modelling, although our models of the ERSoS have not been able to include them due to time constraints.

| N/A |

### SoS Model Verification (SMVA)
SysML does provide tools for development of use cases. We find that use cases in particular are difficult to link to other models, due to very different way of partitioning the system up in a use case when compared to, e.g., block diagrams or activity diagrams. We have not had adequate time to fully explore the possibilities of simulating a use case, however. Further guidance on how to develop use cases appropriate for SoS and then to simulate them would be appropriate.

| 2 |

### SoS Change Propagation Analysis (SCPA)
Identifying specific information exchanged along interfaces in this case study is not always straightforward, even when using graphical notations like SysML, because many messages exchanged are human speech. For this reason, we find it time-consuming and difficult to identify all the occasions on this type of model where the interfaces will be changed, perhaps very subtly, by changes to a constituent system. Some changes which might affect a constituent system’s behaviour can only be identified, for example, by manually comparing the data exchanged with the sequence in which it is exchanged and how that might affect the different possible system states. An added difficulty is that SysML permits only limited automated checking of the constraints or invariants that constituents would like to place on each others’ behaviour, and a lack of a precise way to express these. Despite this, if it’s possible to identify interfaces that have altered as a result of changes to a constituent system, then this would narrow down the code that may need to be re-visited and possibly changed. Whether this narrows it down to any substantial degree will depend on how much is known about the internal organisation of the constituent system in question.

| 2 |

### SoS Interfaces Analysis (SIAA)
We found SysML to be an intuitive tool for identifying and modelling the interfaces between the constituent systems in the ERSoS. The graphical notation is easy to follow which greatly aids human comprehension of

| 3 |
possible weaknesses. SysML’s internal block diagrams are capable of representing graphically the relationships between constituent systems, and the interfaces that define those relationships. Studying internal block diagrams and sequence diagrams side by side we found to be quite a helpful process for iteratively building a picture of interactions between the constituent systems and their environment. The collection of nested block diagrams and internal block diagrams allows the capturing of interactions between the environment and the constituents, and internally between constituent systems themselves.

**SoS Emergent Behaviours Requirement (EBRA)**
We find that SysML provides good support generally for requirements modelling with techniques for ensuring traceability. It’s difficult to develop emergent requirements and identify their contributors, however. SysML does not provide great support for specifying SoS-level properties or emergent behaviour, and analysing how changes to the interface or the sequence of processes will affect some non-functional property or a compositional emergent behaviour which exists at the SoS-level only will be affected. Although SysML does support requirements modelling, it has very little way to express those requirements in a measurable way or to express how a change to one contributor is likely to affect the overall behaviour. This may be a question of lack of guidance being available.

**SoS Fault Tolerance Analysis (SFTA)**
Two of the problem challenges modelled in this deliverable include some fault tolerant behaviour. We find that the SysML sequence diagrams are quite intuitive and allow for relatively easy modelling of the alternative recovery sequence, including identifying the moment when a decision must be made to switch to an alternative, and, importantly, the sequence that should be followed when the alternative system is selected. In this problem domain, it’s crucial to be able to identify problems quickly and to have very clear procedures and sequences in place for moving to alternatives. This greatly aids emergency personnel who are acting under pressure in very fast-developing scenarios where speed is of keen importance. Clear procedures are needed for traceability in a publicly-funded body and for removing sources of confusion amongst human participants on the ground. The models we have developed here do not necessarily make it easy to identify the external events and states that trigger faults, or whether the system will reach the required system state if it follows the recovery strategy rather than following “normal” behaviour. Other SysML techniques that we have not had time to explore thoroughly may aid with this, however, as would greater guidance on the area of SoS fault tolerance.

**SoS Test Model Development (STMA)**
SysML has many advantages, such as providing a very clear and usable graphical user interface. However, it is not formal enough to generate
executable models which would allow the testing of emergent behaviours affected by an altered constituent system, or the delivery of SoS-level non-functional properties.

**SoS Model Terminology Consistency (SMTA)**
The Artisan Studio tool does allow us to ensure that certain models use names (e.g., names of blocks on block diagrams, internal block diagrams, and sequence diagrams) consistently, which does aid in comprehension. Where large numbers of models and diagrams are generated inconsistent naming conventions arise easily and cause great confusion. However, SysML diagrams are not fully formal and there is still some ambiguity possible.

### 5.6 Approach to VDM modelling

The SysML model of the ERS allows us to semi-formally describe the structure and behaviour of the SoS at various levels of abstraction. Translating key features of this to VDM allows us to define more precisely the behaviour of the constituent systems and their interactions. Pre-conditions allow us to define which states an operation should (and should not) be called from if it is expected to deliver the result given in the post-condition. All of the SysML diagrams were examined before creating the VDM model, but the internal block diagrams (e.g. Figure 4) and activity diagrams (e.g. Figure 6) were particularly useful as these show the interfaces and the behaviour of the SoS respectively.

The intention of the VDM model is to define the behaviour of the SoS as a whole, not the structure of Insie’s call centre. Therefore, whilst the SysML model defines the CUS to include separate systems such as the Web Server, Client and Operators, we make suitable abstractions in the VDM model, providing one class to encompass the whole of the CUS. Otherwise the classes in the VDM model correspond to those constituent systems defined by blocks in the SysML model. The interfaces in the SysML model (except those defining interactions within the CUS) were, where appropriate, mapped to public operations in the VDM model. The possible sequences of operation calls both within the CUS and between constituent systems were derived from activity diagrams such as Figure 6.

The VDM model does not describe all aspects of the SysML model, but instead focuses on the functionality required for answering and responding to a call requesting an emergency response. Some constituent systems, such as the Map Provider and the Hospital are, at this stage, abstracted away from.

#### 5.6.1 VDM Model

The constituent systems of the SoS are defined as VDM classes, and as such the VDM Model consists of the following primary classes:
These classes contain the main functionality of the VDM model of the SoS. A number of other classes were defined for structuring purposes. The *Insiel* class (not given in the SysML model) provides a logical grouping of the *PhoneSystem* and *CUS* as these are both managed by Insiel (though the *PhoneSystem* is provided by an external organisation and may vary between call centre buildings) in a collection of call centres. Two abstract classes (*Communicator* and *RadioReceiver*) have been defined to allow generic interactions between the *TeleCom* and *RadioSystem* classes and their users. These are, therefore, sub-classed by users of the *TeleCom* and *RadioSystem* respectively. Skeleton classes have also been defined for two further constituent systems (*Hospital* and *MapProvider*) that are identified in the SysML model, but have not been fully modelled in the VDM at this stage, but form a place of future work if required. The *SoS* class sets up the constituent systems of the ERS SoS.

A separate class (*Environment*) is also created to model the environment of the SoS. This currently only consists of a set of *Caller* objects (to model the behaviour of the public calling the emergency services), but could also include the target locations for providing aid and traffic information in a more detailed model of the SoS.

Finally a class *TestClass* has been outlined that sets up some initial values for constituent systems of the SoS. However, as some of the functionality is defined implicitly (or in the case of the ERU, for example, only in terms of the interfaces between systems), the model cannot be executed in its current state. To allow us to fully execute and test the model using the test facilities provided by VDM++, the model will require further effort to translate to explicitly defined operations.

In the remainder of this section we describe the behaviour involved in answering and processing an emergency call and illustrate it with appropriate excerpts from the VDM model. The complete model is available in Annex 1.

The process of receiving and handling an emergency call can be summarised in a number of steps:

- A Caller dials the emergency response number and the call is placed in a queue until an operator is available to process it. This is described in Section 4.1.1.1.
- Section 4.1.1.2 details the CUS (in the SysML model this is performed by the CUS Operator, abstracted from this model) picking up the call from the queue and requests information from the Caller until sufficient information is known to process the emergency response.
In Section 4.1.1.3, the CUS selects an appropriate set of ERUs for the emergency response (diverting ERUs from less critical responses if necessary) and sends the chosen ERUs the response data.

The ERU services the response, updating the CUS with its status at various stages until the response is complete. This is detailed in Section 4.1.1.4.

4.1.1.1 Placing a call

A Caller initiates and emergency response by placing a call with the TeleCom, using the placeCall(...) operation of the Caller class:

```java
public placeCall : TeleCom\PhoneNumber ==> bool
placeCall(d) ==
    (dcl call : [TeleCom\CallId] := telecom.placeCall(d, self);
     return call <> nil)
-- Assumes the caller is not using their phoneline
pre callInProgress = nil and telecom.isValidPhoneNumber(d);
```

The precondition of the operation (callInProgress = nil) ensures a call cannot be placed if the caller already has a call in progress. The call is successful if the number dialled has a free phone line (indicated by the TeleCom returning a valid call identifier instead of nil). If a call is placed, the caller assigns the call identifier to its local variable callInProgress.

The TeleCom class contains a mapping of PhoneNumbers to Communicators (the abstract class for systems which may place telephone calls), which acts as a directory. The placeCall(...) operation of the TeleCom ensures that the destination (d) has a free phone line (communicators(d).freePhoneLine()), and if so, creates a new call identifier and initiates the call for the destination communicator (communicators(d).initiateCall(call)). The call identifier is returned to the operation caller.

```java
public placeCall: PhoneNumber * Communicator ==> [CallId]
placeCall(d, s) ==
    --if the destination has a free phone line...
    if communicators(d).freePhoneLine()
    then
        (dcl call : CallId := generateNewID();
         currentCalls := currentCalls munion {call |->
             mk_TelcoCall(communicators(d), s, getTime())};

         s.initiateCall(call);
         communicators(d).initiateCall(call);
         return call )
    --if the destination does not have a free phone line...
    else return nil
    pre s in set rng communicators and
        d in set dom communicators and s.freePhoneLine();
```
When a caller places a call to the emergency services, the TeleCom calls the initiate call of the PhoneSystem of the Insiel class. This operation simply places the new call identifier into a queue of calls until the CUS is available to handle it:

```java
public initiateCall : TeleCom`CallId ==> ()
initiateCall(cid) ==
{
    queue := queue ^ [cid];
    currentCalls := currentCalls union {cid}
}
```

Note that the interfaces between the Caller, TeleCom and PhoneSystem are not defined in the SysML model as this focuses on the interfaces between the CUS and the other constituent systems. Therefore a set of sensible interfaces and operations were assumed based on the defined behaviour between the CUS and the PhoneSystem.

### 4.1.1.2 Answering and processing an emergency call

The bulk of the modelling effort when developing the VDM model was given in the CUS class for the purposes of answering and processing an emergency call. The majority of this functionality is given as internal private operations. When the CUS is able to take a call, they can answer the emergency call as follows:

```java
public answerCall : () ==> ()
answerCall() ==
{
    dcl newCall : TeleCom`CallId := ps.answerCall();
    dcl respId : ResponseId := generateNewID();
    processCall(newCall, respId)
}
```

The operation body first invokes the `answerCall()` operation of the PhoneSystem (`ps`), equivalent to the “answer call from queue” operation in the SysML “IIF CT Phone conversation handling” interface. The operation subsequently calls the private `processCall` operation of the CUS.

The Caller is then asked for the details of the emergency using various private operations in the CUS. For example, if the call is determined to be about a new mission the priority of this mission is then assessed:

```java
private handleNewResponse : ResponseDetails * TeleCom`CallId * ResponseId==> ()
handleNewResponse(r, c, respId) ==
{
    --if ERU response is needed
    if r.requiredERUs > 0
        then
            (}
```
It is possible that in assessing the priority of the mission it is determined that no response is needed (obtainPriority(r) returns nil), in which case the call is ended. If this is not the case then the response details are recorded and the response is initiated. This functionality was derived from the “Process Call” activity diagram in the SysML model.

### 4.1.1.3 Initiating the emergency response

Once it has been established that ERUs are required to respond to the emergency call, the CUS must identify a suitable set of ERUs to service the response. This functionality is provided in the `startResponse` operation. Identifying an available ERU may be straightforward if there are sufficient idle ERUs available (idleERUs <> {}), but ERUs may also need to be diverted from less critical missions.

```haskell
private startResponse: ResponseId ==> ()
startResponse(r) ==
{
  dcl idleERUs : set of ERU'ERUId;
  dcl e: ERU'ERUId;
  dcl divertableERUs : set of ERU'ERUId;
  dcl eruToDivert: ERU'ERUId;
  dcl oldRespId : ResponseId;
  dcl numAllocatedEru : nat := 0;

  --Allocate the required number of ERUs
  while (numAllocatedEru < responses(r).details.requiredERUs) do
  {
    --Find all idle ERUs
    idleERUs:= findIdleERUs();
    --If there are idle ERUs available
    if idleERUs <> {} then
      {e := allocateIdleERU(idleERUs, r);
        radio.sendMessage(radioReceiverId, erus(e).radioId,
                         generateERUMessage(<InitResponse>,r));
      }
    --If there are no idle ERUs
    else
      {
        e := allocateIdleERU(divertableERUs, r);
        radio.sendMessage(radioReceiverId, erus(e).radioId,
                          generateERUMessage(<InitResponse>,r));
      }
  }
```

---

D41.1 – Accident Response Engineering  
Analysis Report Using Current Methods & Tools
In the case an ERU is diverted, the response mission which was previously being serviced by that ERU must be restarted (startResponse(oldRespId)). The functionality shown here was derived from the “Initiate Mission” activity diagram in the SysML model.

Finally, a message is sent to the ERU chosen for the new response via the RadioSystem to request them to attend the target location and provide them with any other details they need. The operation (radio.sendMessage(…)) for sending a message over the radio is based on the “transmit audio communication” operation given in the “IIF voice” interface of the SysML model.

4.1.1.4 Servicing the emergency response
The behaviour of the ERU is modelled at an abstract level, purely in terms of the interfaces of the ERU. Once an emergency response mission has been sent to an ERU it will drive to the location of the target (not modelled) and update the CUS on its progress at key points, for example when it reaches the target location.

The ERU sends status updates such as these through the sendMessage(s, r, msg) operation of its ERURadioReceiver and they are, in turn, processed by the CUS using the public receiveRadioMessage operation:

```plaintext
public receiveRadioMessage : RadioReceiver\RadioReceiverId * RadioSystem\RadioMessage ==> ()
receiveRadioMessage(send, msg) ==
{
  dcl radStatus: ERU\ERUStatus:= convertMsgToStatus(msg);
  dcl eruResponse : ResponseId:= findERUResponse(send);

  erus := erus ++ {send |-> mu(erus(send), status |-> radStatus)};
  --If all ERUs on the reposne are returning or idle...
  if (forall e in set (responses(eruResponse).erus) &
      erus(e).status = <RETURNING> or
      erus(e).status = <IDLE>)
  --then mark the response as complete
  then responses := responses ++ {eruResponse |->
      mu(responses(eruResponse),
      responseStatus |-> <Complete>)};
}
```
The CUS updates the status of the ERU it received the message from. If all of the
ERUs servicing that mission are either <RETURNING> or <IDLE> as a result of
that update the mission status is set to <Complete>. This functionality was
derived from the "Monitor Response" activity diagram in the SysML model. At
present, the model only processes messages from the ERU indicating the
completion of a response. The SysML model indicates other statuses that have
not been considered in the current VDM model.

5.7 Challenge Problem Analysis in VDM

In this section we return to our challenge problems and consider how our
modelling efforts help us address the problems they present. We do not include
completed models for all challenges but where this is the case we consider what
changes would be needed for existing models.

5.7.1 Load modelling

This challenge addresses Insiel’s interest in modelling the run-time usage of the
SoS. In this particular example, we assume that funding for the ERSoS is to be
split between the participating individual emergency services (e.g., the fire
brigade, the ambulance service, alpine rescue, etc.) based on their use of the
system.

This type of activity requires a clear idea of the resources that system works with
and how they are employed, together with the relative involvement of each
constituent system. VDM can be helpful here. It requires identification and
precise definition of all the involved data types, and instances can be created to
represent real resources. The current state (represented by values assigned to
these instances) through the system can help us understand how the system is
used, by whom and to what extent resources are deployed by each partner. This
requires some changes to the existing model. For example, currently the ERSoS
VDM model includes only ERUs in terms of major resources, but a model capable
of monitoring resource usage should define data types for all major ERSoS
resources, such as helicopters, telecomms and vehicles. In addition, a way must
be found to store the necessary details for billing each incident or rescue
attempt. Currently the CUS class in the ERSoS VDM model has a variable
Response, which stores some information about the incident and the associated
resources deployed; this variable could be extended to include extra the
information.

A new class is needed (e.g., UsageMonitor) to actually monitor the usage of these
resources, and consideration must be given to how it will interace with the rest
of the SoS. There is a possibility that the architecture of the SoS (currently a
centralised, “star” topography) may need to change to accommodate the
monitoring. For example, it’s possible that the monitor may need to introduce
new interfaces to gather data from each constituent system separately, or it may
simply need to monitor the CUS alone, which requires few changes to the
architecture.
Externally to the model an agreement must be reached on how to attribute use of a resource. For example, if one incident requires the attendance of three emergency services, how will costs and resources be allocated? Some requirements modelling techniques should be employed to negotiate with constituent systems and to model the costs and benefits for each. This particular challenge problem introduces the problem of trust, a widely-acknowledged problem for many SoSs. In this case, the constituent systems who will now be accepting responsibility and charges for use of the ERSoS need to have trust that the bill is correctly and fairly calculated. VDM could be helpful for modelling the charging rules and negotiating these with constituent systems. There is a possibility that the behaviour of constituent systems will alter if the new rules are introduced, with the goal of lowering running costs under the new charging regime. A simulation in VDM may even help to identify the possible changes.

5.7.2 Evolution of a constituent system

Challenge B is concerned with addressing the issues that are raised if a constituent system proposes some change. Insiel need to understand quickly how that change impacts on other constituent systems, and, if possible, to map those changes to a costing model that the cost of making the change can be ascertained. As an example, consider the case that ERUs carry updated radio “datcom” equipment that can communicate current location to the CUS automatically. This is not possible with current equipment, so at the moment ERUs explain their location to the CUS operator over the radio, and the operator updates the CUS manually.

To reflect the current situation, the ERSoS VDM model incorporates a notion of the ERU’s status, which is represented by the ERUStatus in the ERU class. The ERU class also contains an instance of ERURadioReceiver, which has a sendMessage operation for transmission of the appropriate radio messages.

There are several ways to implement the proposed change with the upgraded radio equipment that is capable of communicating location directly to the CUS. To start with, we propose creating a new class, DatCom, to represent the new “datcom” radio device that is capable of transmitting structured, machine-readable radio messages of rich data (such as location, medical records etc). This adds some “future-proofing” to the system, by creating a class that can be extended easily in future if the decision is taken to transmit further rich data via this new radio device in addition to location. The new DatCom class should inherit from the RadioReceiver class; this ensures that it inherits all the appropriate functionality associated with radio transmissions and can be substituted where necessary for other radio devices that also implement RadioReceiver. The ERU class will need to be extended with a new datatype, (e.g., ERUlocation). This takes the form of GPS co-ordinates and will represent the current ERU location as read from some GPS-enabled device.

At this stage consideration must be given to how the new DatCom class will communicate with the CUS. This is a new interface to be introduced, because currently the location is broadcast over the standard voice channel and therefore reuses the standard “voice” radio interface, represented simply as unstructured
data in the current model. We do not wish to replace the existing interface, because unstructured radio communications are still required for other purposes even if location is automatically transmitted. VDM does not directly represent the notion of an interface; instead, it has the notion of operations in classes, and of communications from one class to another. We can accommodate the new interface, therefore, by adding extra operations to the `CUSRadioReceiver` class to accommodate the new type of radio transmission from the `DatCom`. Modelling this side-by-side with a SysML model may be useful for visualising the new interface.

In Section 3.4 the key questions to be answered for Challenge B included analysing the impact of the change, on constituent systems and interfaces, and also studying the SoS non-functional requirements. VDM does help to identify changes to interfaces, although this is quite indirect because VDM does not directly represent an interface as such. However, we can begin to extend the model with new classes and operations, and the changes to the data that is exchanged between classes can then be extracted from the new model. We have found that VDM modelling is best matched with SysML modelling, as the presence of a new interface is easily modelled using graphical notations in SysML, and then studied more carefully in VDM to ascertain the specific changes to data.

Although VDM models the state (the current values held throughout the system) and shows how they change over time, it does not make it easy for us to model the transitions between states and the sequences in which changes happen. This makes it difficult to determine effects of changes on sequences of events. However, we might expect some changes in protocols and sequences to arise if we change the nature of radio communications. We find that modelling radio transmissions in VDM does present some difficulties, particularly when dealing with large numbers of distributed ERUs who may be attempting to communicate at the same time, because concurrency is difficult to represent in VDM. We could rely on SysML for modelling this functionality, although it’s often not clear how to ensure that the two models are consistent with each other, that assumptions and findings from each model are properly disseminated and that the models represent the system in a similar way (so that findings from one model can be employed in another, for example).

Finally, we find it difficult to consider how a change like this would affect SoS level non-functional requirements. Although we can develop and represent requirements in SysML models, VDM does not have a notion of requirements management. We find it difficult to express a performance requirement that is contributed to by multiple constituent systems. We leave this to the SysML model (this example is discussed with the SysML model in Section 5.3.1).

### 5.7.3 A changed environment

Challenge C (introduced in Section 3.4) considered the questions that arise if the ERSoS is to be installed in a new region. Many of our observations and findings for the previous challenge also apply here, because the changed environment
implies that there will be new or evolved versions of constituent systems and/or new interfaces to which the ERSoS must adapt. For example, we expect that a different region will have different devices carried by ERUs and therefore slightly different functionality and radio protocols. The methods described in the previous challenges would therefore be adopted to identify where interfaces are different and to model the changes that are required. SysML models for this challenge are discussed in Section 5.3.2).

5.7.4 Adding a new constituent system that changes the architecture

We consider as an example the case that ERUs are issued with mobile phones as a backup communication method. This would necessitate adding a new constituent system to the existing VDM model. We could accomplish this by adding a new class, Mobile Phone, to the existing model. The existing ERSoS model makes strong use of abstract classes, which is helpful for adding a new constituent system, as inheritance can be leveraged to create a new class that inherits basic functionality already defined elsewhere in the model. Specialised behaviour is then added to the child class as appropriate. In this case the new Mobile Phone class should inherit from Communicators (the abstract class for systems which may place telephone calls). This ensures that all classes used as communicators produce some consistent behaviour, and it allows us to substitute one for the other, as we can simply refer to a class as a Communicator and make no assumptions about whether it is a mobile phone or a radio that the ERU is currently using.

The description of Challenge D in Section 3.4 pointed out the need to identify changes that may be required for other constituent systems, or new interfaces that may be available. Adding a brand new class that inherits from an existing class does immediately bring to our attention to occasions where there are some unacceptable violations of behaviour in terms of the interfaces; if there are operations missing from Mobile Phone that are required of all Communicators, for example, or differences between operations in Mobile Phone and in Communicator, then this is drawn to our attention. There may be new interfaces that are only relevant for Mobile Phone, however, and VDM does not provide very much support for us to automatically identify and add these – they must be identified by hand. SysML’s graphical notation is perhaps a more intuitive tool for exploring possible new interfaces internally within the SoS or externally with the environment, so we might consider iteratively developing the new class using both SysML models and VDM. New interfaces that are available due to the new class/constituent system are easy to identify, now, however; these will be the public operations in Mobile Phone.

One aspect of this challenge that is quite difficult, however, is bringing together the two types of model created by SysML and VDM. The two models have different strengths and weaknesses but it can be difficult to know when to turn from one model to the other. More guidance in this area would be helpful.
5.8 VDM Modelling Evaluation

This section presents some general observations based on Insiel’s experience of modelling the ERSoS in VDM.

Choosing appropriate abstractions

The SysML model is primarily focussed on the structure and functionality of the CUS, plus its interfaces to the other constituent systems. In the VDM model we choose to abstract away from the detailed description of the CUS and considered it as a single system. The intention was to model the behaviour of the whole SoS at the same level of abstraction. Modelling the SoS at this level of abstraction allows us to understand the functionality of the whole SoS without complex implementation details. The CUS VDM class, therefore, does not contain explicit reference to its internal parts: the Web Server, Clients and Operators. The ERU model is also very simple at the moment and does not consider different types of ERU and equipment. If analysis of a more detailed model is required, a model that expands on the implementation details of the CUS could be created at a later date. This more refined model can be compared to the abstract model either through simulation or formal proof.

Maintaining correspondence with the SysML model

The SysML and VDM models were developed in parallel, resulting in a challenge to ensure consistency between the two models. The result of this is that there are some aspects of the case study that are more fully described in the SysML model and some aspects that are more fully described in the VDM model. For example the VDM model describes the interaction between Insiel’s Phone System and the TeleCom in greater detail than the SysML model does, but the SysML model includes information about constituent systems such as the Map Provider and the Hospital that are not included in the VDM model. This is essentially due to the fact that two different groups created the models.

Producing analysable/ executable models

VDM is very good at describing what a system can do without defining how it will be achieved by using implicit functions/operations (pre- and post-condition pairs). Implicit functions were used for some of the CUS functionality (for example in processing the input from a caller to determine the details of an emergency such as its location and criticality). This is a good abstraction mechanism in formal methods, as implicit functions give the ability to state what the result of operation is without placing any implementation restrictions. However, a model containing such functions cannot be executed (although the pre- and post-conditions can be exercised for given inputs and outputs, the outputs cannot be computed from the inputs). Without having an executable model, simulations or combinatorial tests cannot be run to validate the model. Therefore the VDM model produced is not currently executable, but if a more concrete model were created in future work, this issue could be resolved.
On a related point, VDM has support for concurrency, but it was decided that these features should not be included in the abstract model. An avenue for further work therefore would include the development of a model with support for threading to enable multiple ERUs and CUS objects to execute concurrently. This would represent the current ERS more accurately.

5.9 Benchmarking VDM modelling

This section presents benchmarked evaluations of the VDM modelling techniques, based on Insiel’s experiences of creating a model of the ERSoS and considering the four challenges A-D. The benchmarks are allocated a value from 0-4. Benchmarks are explained in detail in D11.2.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SoS Model Verification (SMVA)</td>
<td>2</td>
</tr>
<tr>
<td>VDM is an excellent tool for verification. We do not explore this at length in this deliverable, but we already find that, for example, the precision of VDM allows us to detect where there are erroneous assumptions, operations which might invalidate some conditions or inconsistencies between the data expected/transmitted between two constituent systems. However, we find it very difficult to relate this to use cases. This is partly because VDM and use cases represent the SoS very differently and at different levels of abstraction.</td>
<td></td>
</tr>
<tr>
<td>SoS Test Model Development (STMA)</td>
<td>N/A</td>
</tr>
<tr>
<td>We have not fully explored the capability of VDM to help us create a test model yet.</td>
<td></td>
</tr>
<tr>
<td>SoS State Transition Relation (STRA)</td>
<td>2</td>
</tr>
<tr>
<td>VDM does model state, which is represented as precisely defined datatypes and their current values. We can model states that change over time, but the sequences in which this occurs is difficult to express precisely (it can be achieved, for example, by calling one operation from within another operation, but this complicates analysis). VDM does not have a way for us to model easily the transitions between states, which would be useful for ensuring that all constituent systems are synchronised in their activities. This is important in Insiel’s domain, where human actors distributed around a region need to communicate effectively and in parallel.</td>
<td></td>
</tr>
<tr>
<td>SoS Model Terminology Consistency (SMTA)</td>
<td>3</td>
</tr>
<tr>
<td>VDM is an excellent tool for enforcing consistency. As a formal language it provides precise definitions for its syntax. Identified data types, assumptions and pre- and post-conditions relevant to the system state are all required to be stated explicitly in VDM’s unambiguous syntax. The only difficulty we face is in making sure that the SysML models and</td>
<td></td>
</tr>
</tbody>
</table>
VDM models represent the ERSoS systems in similar ways, so that we can amalgamate useful findings from each model.

**SoS Change Propagation Analysis (SCPA)**

VDM supports a design by contract paradigm, and also allows for dependencies, assumptions and pre- and post-conditions to be stated explicitly. We find that creating the models does help with the elicitation and development of the assumptions and pre- and post-conditions. Once changes have been made to one class, it is possible to see by running the model if other classes are affected. For example, changes that result in the violation of pre- or post-conditions will be identified quickly, as will those changes that have modified the signatures of public methods.

However, some changes that are significant may not be captured easily in VDM, such as sequences of activities. It's not easy to see if one constituent system affected by a change will in turn propagate changes onto others (i.e., the in-bound interfaces are changed, and in turn this results in a change on the out-bound interfaces). VDM allows the representation of some internal processes and internal state of individual constituent systems, which does allow the automatic detection of changes that might propagate in this way. The success rate of a change propagation study of a VDM model of course depends heavily on what is known about each constituent system – if it is a black box we cannot make reliable predictions about its likelihood of propagating incoming changes further onto its neighbours.

**SoS Interfaces Analysis (SIAA)**

Unlike SysML, VDM does not have a notion of an interface. Instead, this information can be gathered by studying the public operations of a class. Without a visual interface to help (as found in SysML) the VDM model can become confusing if many classes interact together – it's easy to lose track in a complex model of where there interactions between different classes. Clear design guidelines are important here. However, VDM is immensely useful for helping to refine the interfaces which are identified in SysML initially, by ensuring that data to be exchanged is refined and stated explicitly.

**SoS Model Identification (SMIA)**

VDM is a useful tool for modelling some aspects of an SoS. It captures precise requirements for data requirements and interfaces between constituent systems, for example. It also allows for precise expression of assumptions and constraints made by constituent systems and/or the environment, and allows the detection of “violating” operations that would leave the system in an unacceptable state (via pre- and post-conditions). This requires effort initially to ensure that assumptions, conditions and constraints are elicited and captured as required. However, it does not necessarily contribute towards the identification of
architectural elements and patterns in a user-friendly way, and it does not support the notion of an architecture, although it is possible that a VDM model could be used in conjunction with another model (e.g., a SysML model) to explore some aspects of performance. We feel that more guidelines about SoS architectural patterns are needed in this area.

<table>
<thead>
<tr>
<th>SoS Architectural Modelling (SDAA)</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDM is not really intended as tool for supporting architectural analysis. It does not provide any system views to aid comprehension or exploration of system topography and it does not encompass a explicit notion of interfaces between constituents (or subs-systems). It may allow some exploration of the performance trade-offs of different organisations of the system, however.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SoS Emergent Behaviours Requirement (EBRA)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>VDM does not support the concept of a requirement, and we find that it is not always straightforward to ensure traceability between a VDM model and a requirements model, in part because these tend to represent very different ways of visualising the SoS. The formal properties of VDM may allow some exploration of how functionality should be partitioned between constituent systems and how far each constituent system may contribute towards an emergent SoS requirement, however. It may be very difficult to model the contribution towards non-functional requirements such as security and performance, however, because we have found it quite difficult to model concurrency and sequencing in VDM. Security and performance (e.g., speed of response) are very important non-functional requirements in Insiel's domain.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SoS Fault Tolerance Analysis (SFTA)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time spent modelling and eliciting conditions, assumptions and constraints for a VDM model can form part of an elicitation process for identifying faults that may arise in the system, and how and where they may be detected. It's also possible to add functionality to a model to represent an alternative set of behaviours that should be followed when some fault is detected, and to ensure that operations which would leave the system in an unacceptable state are not executed. However, if there are a large number of faulty scenarios to explore the model may become very complex very quickly. It may not be straightforward to visualise the propagation of a fault across the constituent systems, and without a detailed ability to model and analyse sequencing there will be many faults and errors that are difficult to detect in a VDM model.</td>
<td></td>
</tr>
</tbody>
</table>
5.10 Challenges & Modelling Decisions

The SysML and VDM modelling tasks presented a number of challenges. The key challenges are summarised in the following sections, along with the modelling decisions that were made. We make observations on the baseline notations: SysML in Section Error! Reference source not found. and VDM in Section Error! Reference source not found.. In Section Error! Reference source not found. we evaluate the modelling effort with respect to the initial objectives.

Given the SysML and VDM models of the emergency response SoS, we may draw conclusions on the modelling effort with respect to the objectives described in Section 3.3, and we may consider how useful the SysML and VDM modelling exercises have been for modelling the ERSoS.

A large portion of the effort in VDM modelling was spent in eliciting details of the relevant constituent systems and in learning how to produce the large-scale SysML model presented in this deliverable. As such, many of these objectives remain unfulfilled. This is not an indication of the quality of the models, rather the additional effort required to learn how to use the notation given the time constraints.

During the modelling of the case study, we considered the use of the two communication systems between the Call Centre “CUS” and ERU systems. The initial model relied solely upon the Radio System constituent system. As described in Section 3.6.1, we also consider the use of the Mobile Phone system in conditions where the Radio System is unavailable. Whilst this is not a planned change for the time being, the modelling effort allowed us to understand the change in behaviour (seen in the sequence diagrams in full SysML model in Annex 12.3). In future work, we may consider other modifications such as replacement of Telecom systems, or a distributed call centre.

In conjunction with the COMPASS task T2.4.2, we investigated the analysis of faults and fault tolerance in the Emergency Response SoS. This work was largely carried out as part of T2.4.2, and as such will be reported in work undertaken by that task. In the next stage of work, in task T4.1.2, we shall aim to take advantage of COMPASS methods to further investigate faults and fault tolerance in the emergency response case study.

The SysML effort in the behaviour descriptions was limited to activity diagrams and sequence diagrams to explore the flow of control between the constituent systems so to perform the high-level SoS functionality. As the SysML was not executable, we were unable to identify bottlenecks in the performance and efficiency of the SoS – as identified as an objective in Section 3.3. As discussed in Section Error! Reference source not found. elements of the VDM model are not executable, and thus additional effort is required to achieve this objective.

Through the identification of provided and required interfaces between the constituent systems and between the SoS and its environment, we have begun the process of identifying the dependencies between the different SoS elements. The interfaces are defined as simple operation signatures in the SysML models,
and enriched with pre/post conditions and explicit operation specifications in the VDM model. Whilst the identification of the interfaces in the SysML model has helped in recording the dependencies, the model is limited in its ability to communicate the protocols at these places of interaction. A robust and systematic approach to interface specification should be taken in future modelling efforts to address this issue. The use of SysML requirements diagrams and traceability relations may also aid in understanding the relationships between the points of interaction and external requirements describing dependencies.
6 Conclusions

In task T4.1.1 we have examined both VDM and SysML approaches to improving modelling and analysis of the emergency response case study. The preceding chapters have described the case study from a functional point of view and modelled this case using the baseline technologies available in Insiel during Phase 2 of the project. The document has described the objectives that are important for industrial stakeholders like Insiel and the challenges that arise in the public emergency management field for system of system engineering. It details how this case study fits into the SoS landscape and characterises the SoS itself. As we move to Phase 4 of the project we can demonstrate whether the COMPASS tools have delivered the desired improvements over current practice.

This deliverable documents the objectives and SoS modelling needs that drive the case study supplied by Insiel. Our initial attempts to model the ERSoS in baseline technologies has demonstrated the areas where we believe improvements still need to be made. In particular, we hope that future work will:

- Improve the ability to document, analyse and implement changes across an SoS to reduce development and maintenance costs of systems development. In particular this feature would support a developer like Insiel to analyse cost parameters, which is fundamental to Challenges A and C, for the clear relationship between the documentation process and the ability to estimate and record projected or accrued costs. It also directly supports Challenges B and C in allowing planners and implementers to model and record the impact of changes in a readable, interoperable manner which can be exchanged between constituent system owners and developers.

- Improve the response time where system performance is temporarily and unexpectedly degraded because an external party has deployed some change without warning.

- Provide engineering techniques demonstrating SoS-level properties including its suitability for depicting fault tolerance, security and emergent behaviour improving mapping non-functional requirements. This characteristic would support the development of new system topographies as constituent systems evolve. This is important for Challenges B and D for describing parameters like security, or the business process impact where new functionalities are required.

- Optimise SoS architecture and behaviour to identify potential bottlenecks where multisystem integrations come together in a complex SoS, like the ERSoS. This is demonstrated in Challenge B, which highlights how often the apparently simple modifications can create bottlenecks. Tools are needed to help identify these bottlenecks at modelling time.

- Improving the SoS performance and supporting the evaluation of different architectural configurations. This would improve our ability to adapt the system to different environments or requirements through architectural and behavioural analysis. Challenge C demonstrates this; in
In this situation, we should like to be able to analyse different configurations for performance before actually deploying code.

- **Improve interaction with stakeholders** and take proactive measures to avoid problems, and document them where they do occur.

Modelling these challenges in VDM highlights the rigid syntactical formalisms, which must be understood prior to using them in modelling. It requires specific training (which has been made available) to understand how to construct and validating a model with the baseline tools. VDM does have a high level of expressiveness and is suited to creating detailed models of non-standard behaviours. In this sense it is practical and responds to real life testing challenges.

However, the baseline VDM tools used in Phase 2 are not easily deployed in industry. SysML is closer to a market-deployed tool. It uses well-known graphical formalisms and is therefore more intuitive. Despite this, because it is quite preconfigured, it tends to be suited to standard process description and requires more effort to provide input in order to obtain results where specific detailed requirements are involved. When SysML tools will be enriched with the CML extensions, we expect to have an expressive tool which is familiar to engineers. Using these tools we believe we can demonstrate a marked improvement in the analysis, documentation and deployment of SoSs in Insiel’s particular market and for the types of integration activities foreseen in the case study. VDM and SysML modelling approaches appear to be effective in highlighting external dependencies to an SoS and in improving our ability to identify constituent systems with an increased risk of unexpected changes propagated from the SoS environment.

We perceive that there is a need to improve the SoS engineering process, by giving analysts the analytical tools and constructs to reduce the engineering process effort by codifying many of the routines which were previously impossible to represent, providing a templates for processes often replicated as well as reducing the overall time dedicated to the engineering process itself.

We have identified a number of challenges and areas from the industrial stakeholder point of view that we expect to be improved. As well as tools, guidelines are needed to support the user in selecting the most appropriate tool in each stage of the analysis. We expect the project to provide routines for developing models with syntax nearer to the CML solving specific domain challenges (like validating the system in case of concurrent access to resources).

At this point in the project we have used the baseline tools made available to model the interactions that are typical in the CUS system, identifying challenges and giving more detail to the objectives for an industrial stakeholder like Insiel. We understand more clearly what characteristics a case study like the CUS system presents to the SoS domain and therefore have been able to better identify which constraints should drive the development of the upcoming phases in the project.
7 References


Annex I VDM Model

Model Overview

In Figure 22 we show the structure of the VDM model. The composition and relationships between constituent systems are identified. The top-level element of the case study is the SoS and it’s constituent systems: the TeleCom, RadioSystem, ERUs and the Insiel system, containing the CUS and the PhoneSystem. The process of responding to an emergency call is described in Section 4.2 of this deliverable.

Figure 22 - Block Definition Diagram showing structure of VDM model

Model Classes

Caller Class

-- Models the person who initiates an emergency response.
-- Inherits the Communicator class to enable a generic
-- interface to the TeleCom
class Caller is subclass of Communicator

instance variables

-- The telecommunications system used by the caller
private telecom : TeleCom;

-- The call identifier of any call currently in progress
-- (can be nil)
private callInProgress: [TeleCom\`CallId];

operations

-- Constructor for the Caller. Inputs the telecommunications
-- system to be used by the caller. A Caller object has
-- exactly one phone line
public Caller : TeleCom ==> Caller
Caller(t) ==
{
    telecom := t;
    callInProgress := nil;
    phoneLines := 1;
    currentCalls := {}
};

-- Models a Caller using their phone to place a call
-- Returns false if the destination has no free phone
-- lines otherwise updates the current calls of the Caller
public placeCall : TeleCom\`PhoneNumber ==> bool
placeCall(d) ==
{
    -- Attempt to place a new call to the destination number
dcl call : [TeleCom\`CallId] := telecom.placeCall(d, self);
    return call <> nil
}
-- Assumes the caller is not using their phoneline
pre callInProgress = nil and telecom.isValidPhoneNumber(d);

-- Initiate the call - adds it to the set of ongoing
-- calls and make the callId the current call in progress
public initiateCall : TeleCom\`CallId ==> ()
initiateCall(call) ==
{
    callInProgress := call;
    currentCalls := currentCalls union {call};
}
-- the call must not be in progress and the caller
-- must have a free phone line
pre call not in set currentCalls and
    card currentCalls < phoneLines
post call in set currentCalls and
    card currentCalls <= phoneLines;

-- End a given call - removes it from the set of ongoing
-- calls and make the current call in progress = nil
public endCall : TeleCom\`CallId ==> ()
endCall(call) ==
{
    callInProgress := nil;
    currentCalls := currentCalls \ {call};
}
-- the call must be in progress in order to end the call
pre call = callInProgress
-- ensure call is no longer in progress
post callInProgress = nil;

--Generates and forwards responses to questions received
--over the phone. Return message may be nil if no response
--is given by the Caller
public converse : Telecom`CallId * Message ==> [Message]
converse(cid, msg) ==
{
  return genResponse(msg)
}
-- Assumes that the call relating to the request is
--currently in progress
pre cid = callInProgress;

-- Generates a response to a request for information
-- We do not specify how the response is generated
private genResponse : Message ==> [Message]
genResponse(msg) == is not yet specified;
end Caller

**Communicator Class**

--Abstract class to enable communications with the Telecom
--class
class Communicator
types
  public Message = token;

instance variables

--variables to signify the phonelines a communicator owns
--and the set of current calls on those phonelines
protected phoneLines : nat;
protected currentCalls : set of Telecom`CallId;
inv card currentCalls <= phoneLines;

operations

-- Constructor - initialises variables
public Communicator : nat ==> Communicator
Communicator(pl) ==
{
  phoneLines := pl;
  currentCalls := {}
};

--Operation to initiate a call
public initiateCall : Telecom`CallId ==> ()
initiateCall(cid) == is subclass responsibility
pre cid not in set currentCalls and card currentCalls < phoneLines
post cid in set currentCalls and card currentCalls <= phoneLines;
-- Operation to converse and exchange messages with
-- another Communicator
public converse : TeleCom`CallId * Message ==> [Message]
converse(cid, msg) == is subclass responsibility
pre cid in set currentCalls
post phoneLines = phoneLines~ and currentCalls = currentCalls~;

-- Operation to end a given call
public endCall : TeleCom`CallId ==> ()
endCall(cid) == is subclass responsibility
pre cid in set currentCalls
post cid not in set currentCalls and
   card currentCalls = card currentCalls~ -1;

-- Operation to determine if there are any free phone lines
public freePhoneLine() res : bool
post res = card currentCalls < phoneLines;

-- Operation to determine if a given call is currently in progress
public isCurrentCall : TeleCom`CallId ==> bool
isCurrentCall(call) ==
   return call in set currentCalls;

end Communicator

CUS Class

-- CUS class represents the complete Call Centre, abstracting
-- from workstations and operators. The CUS takes calls from
-- the PhoneSystem, answers the calls, and generates an
-- appropriate response by allocating ERU resources to the
-- response.
class CUS

types

public Response :: details : [ResponseDetails]
calls : seq of TeleCom`CallId
erus : set of ERU`ERUId
divertedErus : set of ERU`ERUId
responseStatus : <OnGoing>|<Complete>|<Discarded>|<ExtOrg>;

public ResponseDetails :: target : token
criticality : Priority
patientInfo : token
requiredEquipment : token
requiredERUs : nat
extOrgs : [ExtOrg];

public Priority = <UnAssigned>|<Red>|<Yellow>|<Green>|<White>;

public ResponseId = token;

public ExtOrg = <Fire>|<Police>|<ForestRanger>|<CivilProtection>;
public ReqType = <InitEval>|<ObtainCasDetails>|<ObtainFurtherDetails>;

d -- Assume every eru has a radio channel
public ERUComType = <InitResponse>|<UpdateResponse>|<DivertERU>;

values
-- Assume only 10 ERUs to manage.
private numERU : nat = 10;

instance variables

private radio : CUSRadioReceiver;
private radioReceiverId : RadioReceiver`RadioReceiverId;
private ps : PhoneSystem;
private extOrgDir : map ExtOrg to TeleCom`PhoneNumber;
private responses : map ResponseId to Response;

-- The set of all known ERUs (some are idle and some are --busy)
private erus: map ERU`ERUId to ERUInfo;

-- Invariant ensures that the CUS does not manage more ERUs --than the maximum value...
inv card dom erus <= numERU and
-- Any two ongoing responses should have different ERUs --servicing the response and with different patients
(forall r1, r2 in set rng responses &
  (r1 <> r2 and r1.responseStatus = <OnGoing>
  and r2.responseStatus = <OnGoing>) =>
  (r1.erus inter r2.erus = {} and
   r1.details.patientInfo <> r2.details.patientInfo))
-- All ongoing responses should require at least one ERU --but not have more ERUs servicing the response than --required
   and (forall r in set rng responses &
    (r.erus subset dom erus and
     r.responseStatus = <OnGoing>)) =>
     (r.details.requiredERUs > 0 and
      card r.erus <= r.details.requiredERUs))
-- ERUs must have unique channel IDs.
   and (forall e1, e2 in set rng erus & e1 <> e2 =>
     e1.radioId <> e2.radioId);

operations

-- CUS Constructor
public CUS : RadioSystem * PhoneSystem * set of ERU *
  RadioReceiver`RadioReceiverId *
  map ExtOrg to TeleCom`PhoneNumber ==>
  CUS
CUS(rs, p, es, rId, extDir) ==
  { radio := new CUSRadioReceiver(rId, rs, self);
    rs.addChannel(rId, radio);
ps := p;
responses := {\{\} | -> mk_ERUInfo(aru.getRadioId(),
<IDLE>>) | aru in set es};

-- assuming ERUS are idle to begin
radioReceiverId := rId;
extOrgDir := extDir;

--Answer a call from the phonesystem, generate a new
--responseId and begin proccessing the call.
public answerCall : () => ()
answerCall() ==
{
dcl newCall : TeleCom`CallId := ps.answerCall();
dcl respId : ResponseId := generateNewID();
processCall(newCall, respId)
}

--Begin processing a call, determining if need to generate
--a response. Assuming discarded calls is the same as new
--call that does not require response
private processCall : TeleCom`CallId * ResponseId => ()
processCall(call, respId)==
{
--If response is needed
if initialEvaluation(call)
then
  --Process response details
  dcl respDetail : ResponseDetails := obtainDetails(call);
  processResponseDetails(respDetail, call, respId)
}
--If no response needed...
else
  --Log call as discarded and end call
  responses := responses \union
  {respId | -> mk_Response(nil, [call],
  {},{},<Discarded>)};
  ps.endCall(call);
}
pre respId not in set dom responses;

--Determine whether the response is new, existing or for
--external orgs
private processResponseDetails : ResponseDetails *
  TeleCom`CallId * ResponseId => ()
processResponseDetails(respDetail, call, respId) ==
{
--If need to divert call to external org...
if needDivertCall(respDetail)
then

responses := responses munion
  {respId |-> mk_Response(nil,[call],
    {},{},<ExtOrg>});
ps.divertToExternalOrg(call, extOrgDir(respDetail.extOrgs))
)

--Otherwise handle the call
else
  (  
    --First check if call is for a duplicate response
    dcl exRId : [ResponseId] := existingResponse(respDetail);
    if exRId = nil
      then handleNewResponse(respDetail, call, respId)
      else handleDuplicateResponse(respDetail,exRId,call)
  )
)

pre respId not in set dom responses;

--Handle a call which generates a new response
private handleNewResponse : ResponseDetails *
  TeleCom`CallId * ResponseId==> ()
handleNewResponse(r, c, respId) ==
  (  
    --if ERU response is needed
    if r.requiredERUs > 0
      then
        --Check response priority
        dcl respPriority : Priority := obtainPriority(r);
        dcl r2 :ResponseDetails := mu(r, criticality |-> respPriority);
        responses := responses munion
          {respId |-> mk_Response(r2,[c],
            {},{},<OnGoing>});
        startResponse(respId)
      else
        (  
          responses := responses munion
            {respId |-> mk_Response(nil,[c],
              {},{},<Discarded>});
          ps.endCall(c)
        )
  );

--Start a response with ERU
private startResponse: ResponseId ==> ()
startResponse(r) ==
  (  
    dcl idleERUs : set of ERU`ERUId;
    dcl e: ERU`ERUId;
    dcl divertableERUs : set of ERU`ERUId;
    dcl eruToDoDivert: ERU`ERUId;
    dcl oldRespId : ResponseId;
    dcl numAllocatedEru : nat := 0;
  )
Allocate the required number of ERUs

while (numAllocatedEru < responses(r).details.requiredERUs) do
{
    -- Find all idle ERUs
    idleERUs := findIdleERUs();
    -- If there are idle ERUs available
    if idleERUs <> {} then
        ( -- Allocate a given idle ERU and send a
          -- message to that ERU
          e := allocateIdleERU(idleERUs, r);
          -- COMM to ERU via radio system
          radio.sendMessage(radioReceiverId, erus(e).radioId, generateERUMessage(<InitResponse>, r));
        )
    -- If there are no idle ERUs
    else
        ( divertableERUs := divertableERUS(r);
          if divertableERUs <> {} then
            ( -- Obtain a ERU to divert, get local variable
              -- for the response to reallocate
              eruToDivert := chooseERUToDivert(r, divertableERUs);
              oldRespId := reAllocateERU(r, eruToDivert);
              -- COMM to ERU via radio system
              radio.sendMessage(radioReceiverId, erus(eruToDivert).radioId, generateERUMessage(<DivertERU>, r));
              -- Restart the old response
              startResponse(oldRespId);
            )
          )
          -- NOT CURRENTLY MODELLING NO DIVERTABLE ERUs
        );
    numAllocatedEru := numAllocatedEru + 1;
}

-- Operation for receiving messages from ERU
receiveRadioMessage(send, msg) ==
{
    -- Process incoming message and obtain response the
    -- sender ERU is working on
    dcl radStatus: ERU`ERUStatus := convertMsgToStatus(msg);
    dcl eruResponse : ResponseId := findERUResponse(send);
    erus := erus ++ {send |-> mu(erus(send), status |-> radStatus)};
    -- If all ERUs on the response are returning or idle...
    if (forall e in set (responses(erusResponse).erus) &
        erus(e).status = <RETURNING> or
        erus(e).status = <IDLE>)
        -- then mark the response as complete
then responses := responses ++ {eruResponse |->
    mu(responses(eruResponse),
        responseStatus |-> <Complete>)};

--Only modelling behaviour for <IDLE> and <RETURNING>
--messages, may need to model CUS response to other
--status messages.

} pre send in set rng erus;

--Choose an ERU to divert
--(must respect relative priority and other factors)
private chooseERUToDivert(r : ResponseId,
    divERUs : set of ERU`ERUID) eru : ERU`ERUID
ext rd responses: map ResponseId to Response
    rd erus: map ERU`ERUID to ERUInfo
--precondition states the none of the divertable ERUs are
--currently servicing a response which is more critical
---arranged
pre divERUs subset dom erus and
    forall e in set divERUs &
        not exists r1 in set dom responses &
            e in set responses(r1).erus and
            compareCriticalityFunction(responses(r1),
                                         responses(r))
post eru in set divERUs;

-- Change the response an ERU is allocated to
-- Returns the responseId that the ERU was previously
--assigned to
private reAllocateERU : ERU`ERUID * ResponseId ==> ResponseId
reAllocateERU(enu, r) ==
{
    dcl oldRespId : ResponseId := getCurrERUResp(enu);

    -- Remove the ERU from its current response (and record
    --the history in the divertedERUs field)
    responses := responses ++ {oldRespId |->
        mu(responses(oldRespId), erus |->
            responses(oldRespId).erus \ {enu})};

    responses := responses ++ {oldRespId |->
        mu(responses(oldRespId), divertedErus
            |-> responses(oldRespId).divertedErus
            union {enu})};

    -- Add the ERU to the new response
    responses := responses ++ {r |-> mu(responses(r),
            erus |-> responses(r).erus union {enu})};

    return oldRespId;
}

--precondition states that the erus to reallocate must
--exist, and that precondition states the none of those
--ERUs are currently servicing a response which is more
--critical than the new response for which a diversion is
--being arranged
pre eru in set dom erus and
not exists r1 in set dom responses &
eru in set responses(r1).erus and
  compareCriticalityFunction(responses(r1),responses(r))
post eru in set dom erus;

--Get the response Id a given ERU is servicing
private getCurrERUResp(enu : ERU`ERUId) r : ResponseId
ext rd responses: map ResponseId to Response
  rd eru: map ERU`ERUId to ERUInfo
pre eru in set dom erus
post r in set dom responses and
  eru in set responses(r).erus;

--compare new rescue to all ongoing rescues, returns any
--ERUS which are divertable (by priority or other factor)
private divertableERUS(r: ResponseId) divertable : set of ERU`ERUId
ext rd responses: map ResponseId to Response
  rd eru: map ERU`ERUId to ERUInfo
pre r in set dom responses
--postcondition states the none of the divertable ERUs are
--currently servicing a response which is more critical
--than the new response for which a diversion is being
--arranged
post divertable subset dom erus and
  forall e in set divertable &
  not exists r1 in set dom responses &
  e in set responses(r1).erus and
  compareCriticalityFunction(responses(r1),responses(r));

--Retrieve list of all idle ERUs
private findIdleERUs() idleERUs: set of ERU`ERUId
ext rd erus: map ERU`ERUId to ERUInfo
  rd responses: map ResponseId to Response
post idleERUs = dom erus \
  (dunion {r.erus | r in set rng responses});

--Select an idle ERU according to some unspecified criteria
private allocateIdleERU(idles : set of ERU`ERUId,
r: ResponseId) eru: ERU`ERUId
ext rd responses: map ResponseId to Response
  rd eru: map ERU`ERUId to ERUInfo
--idle ERUs must exist and not be servicing any response
pre idles subset dom erus and
  (idles inter (dunion {r1.erus | r1 in set rng responses}) = ())
post eru in set responses(r).erus and eru in set idles;

--handle a duplicate call
private handleDuplicateResponse: ResponseDetails *
  ResponseId * TeleCom`CallId ==> ()
handleDuplicateResponse(r,c,rId) ==

-- Add the call to the response
responses := responses ++ {rId |-> mu(responses(rId),
calls |-> responses(rId).calls ^ [c])};
ps.endCall(c)
);

--returns true if call is genuine, false if not
private initialEvaluation : TeleCom`CallId ==> bool
initialEvaluation(call) ==
{
dcl exchange : seq of Communicator`Message :=
    requestInfo(call, <InitEval>);
return isGenuine(exchange)
};

--returns ResponseDetails for some communication exchange
private obtainDetails : TeleCom`CallId ==> ResponseDetails
obtainDetails(call) ==
{
dcl exchange : seq of Communicator`Message :=
    requestInfo(call, <ObtainCasDetails>);
return getResponseDetails(exchange)
};

--Operation to convert a message obtained from an ERU to a
--ERU status
private convertMsgToStatus : RadioSystem`RadioMessage ==> ERU`ERUStatus
convertMsgToStatus(msg) == is not yet specified;

--Determine if need to divert to an external organisation
private needDivertCall:  ResponseDetails ==> bool
needDivertCall(r) ==
return (r.extOrgs <> nil);

--determine if the response id already ongoing
private existingResponse(respDet : ResponseDetails) rId:
    [ResponseId]
ext rd responses
--Postcondition states that if there is a response with the
--given response Details, then the returned Id matches this
--response, otherwise nil is returned
post (not exists r in set dom responses &
    compareResponse(respDet, responses(r).details)
    => rId = nil) and
(exists r in set dom responses &
    compareResponse(respDet, responses(r).details))
=> (rId in set dom responses and
    compareResponse(respDet,
    responses(rId).details));

--simulates conversation with caller, given some context
--(ReqType). When fully defined, would use sendMessage
private requestInfo : TeleCom`CallId * ReqType ==> seq of Communicator`Message

requestInfo(c, rT) == is not yet specified
pre c in set ps.getCurrCalls()
post RESULT <> [];

private generateERUMessage : ERUComType * ResponseId ==> RadioSystem`RadioMessage

generateERUMessage(com, r) == is not yet specified
pre r in set dom responses;

private findERUResponse(eru : ERU`ERUId) rId: [ResponseId]

pre eru in set dom erus
post rId = nil or
(rId in set dom responses and
responses(rId).responseStatus = <OnGoing> and
eru in set responses(rId).erus);

private generateNewID() id : ResponseId

ext rd responses
post id not in set dom responses;

private obtainPriority :ResponseDetails > Priority
obtainPriority(r) == is not yet specified;

private isGenuine : seq of Communicator`Message -> bool
isGenuine(msgs) == is not yet specified
pre msgs <> [];

private getResponseDetails(msgs: seq of Communicator`Message) ret : ResponseDetails

pre msgs <> []
post ret.criticality = <UnAssigned>;

private compareResponse(r1 : ResponseDetails, r2 : ResponseDetails) ret: bool
post ret = (r1 = r2); -- WILL BECOME A LOT MORE COMPLEX

-- Compares the criticality of two responses
-- True if the first response is more critical
compareCriticalityFunction: Response * Response -> bool
compareCriticalityFunction(r, r2) == is not yet specified;
end CUS

CUSRadioReceiver Class

-- Class to enable radio communications between ERUs and the -- CUS.
class CUSRadioReceiver is subclass of RadioReceiver

instance variables

  private cus : CUS;

operations

  -- CUSRadioReceiver Constructor
  -- Initialises the CUS, its radio channel and the -- RadioSystem
  public CUSRadioReceiver : RadioReceiverId * RadioSystem *
    CUS ==> CUSRadioReceiver

  CUSRadioReceiver(id, rs, c) ==
  {cus := c;
   rId := id;
   radioSys := rs
  };

  -- Sends a radio message from the CUS to an ERU
  public sendMessage : RadioReceiverId * RadioReceiverId *
    RadioSystem`RadioMessage ==> ()

  sendMessage(s, r, msg) ==
  {radioSys.sendMessage(s, r, msg)
  }  
  pre s = rId;

  -- Receives a radio message from an ERU and forwards it to -- the CUS
  public receiveMessage : RadioReceiverId * RadioReceiverId
    * RadioSystem`RadioMessage ==> ()

  receiveMessage(s, r, msg) ==
  {cus.receiveRadioMessage(s, msg)
  }  
  pre r = rId;

end CUSRadioReceiver

Environment Class

-- Environment of the SoS
-- Currently very simple, containing a set of callers.
-- In concrete model also include external orgs, traffic etc.
class Environment

instance variables

callers : set of Caller := {};

operations

--Environment Constructor
--Initialises the Callers
public Environment: set of Caller ==> Environment
Environment(cs) ==
{
callers := cs
}
end Environment

ERU Class

--Class to represent an Emergency Response Unit (ERU). Will
--receive instructions from the CUS and services a response by
--providing aid to a specified target.
--In concrete model also include crew, vehicle, equipment.
class ERU

types

public ERUId = token;
public ERUStatus = <NOT READY>|<IDLE>|<TO TARGET>|
              <ON_TARGET>|<TO_HOSPITAL>|<ON_HOSPITAL>|
              <RETURNING>;

instance variables

-- id and radioReceiverId could be merged -- one to one
--mapping at the moment.
private id : ERUId;
private radio : ERURadioReceiver;
private radioReceiverId : RadioReceiver`RadioReceiverId;
private contacts : set of RadioReceiver`RadioReceiverId;

--Assume the contacts list does not contain the ERU’s own
--radioReceiverId (so that they cannot radio themselves)
inv radioReceiverId not in set contacts;

operations

--ERU Constructor
--Initialises the identifier of the ERU, its radio channel,
--the RadioSystem and the ERU’s contacts
public ERU : ERUId * RadioReceiver`RadioReceiverId *
             RadioSystem * set of RadioReceiver`RadioReceiverId ==> ERU
ERU(i, radId, rad, cons) ==
{
id := i;
radioReceiverId := radId;
radio := new ERURadioReceiver(radioReceiverId, rad, self);
contacts := cons
);

--Placeholder for receiving messages
public receiveRadioMessage : RadioReceiver`RadioReceiverId
                  * RadioSystem`RadioMessage ==> ()
receiveRadioMessage(send, msg) == is not yet specified;

--Returns the identifier of the ERU
public getId: () ==> ERUId
getId() == return id;

--Returns the radio identifier of the ERU
public getRadioId: () ==> RadioReceiver`RadioReceiverId
getRadioId() == return radioReceiverId;

--Returns the ERU’s radio receiver
public getRadioReceiver: () ==> RadioReceiver
getRadioReceiver() == return radio;

--Adds a new contact to the ERU’s contacts list
public addContact : RadioReceiver`RadioReceiverId ==> ()
addContact(rId) ==
  (contacts := contacts union {rId})
  pre rId <> radioReceiverId and rId not in set contacts;

end ERU

**ERURadioReceiver Class**

--Class to enable radio communications between ERUs and the
--CUS.
class ERURadioReceiver is subclass of RadioReceiver

instance variables

  private eru : ERU;

operations

--ERURadioReceiver Constructor
--Initialises the ERU, its radio channel and the
--RadioSystem
public ERURadioReceiver : RadioReceiverId * RadioSystem *
                  ERU ==> ERURadioReceiver
ERURadioReceiver(id,rs,e) ==
  (eru := e;
   rId := id;
   radioSys := rs
);

--Sends a radio message from an ERU to the CUS or another
--ERU
public sendMessage : RadioReceiverId * RadioReceiverId * RadioSystem`RadioMessage ==> ()
sendMessage(s, r, msg) ==
{
  radioSys.sendMessage(s, r, msg)
}
pre s = rId;

-- Receives a radio message from the CUS or another ERU and forwards it to the ERU
public receiveMessage : RadioReceiverId * RadioReceiverId * RadioSystem`RadioMessage ==> ()
receiveMessage(s, r, msg) ==
{
  eru.receiveRadioMessage(s, msg)
}
pre r = rId;
end ERURadioReceiver

Insiel Class

-- Class to model the System(s) operated by Insiel
class Insiel

instance variables

  phoneSys : PhoneSystem;
  cus: CUS;

values

  -- Radio ID for CUS.
  cusRadCh: RadioReceiver`RadioReceiverId = mk_token("CUS");

operations

  -- Constructor to initialise Insiel-operated systems.
  -- Requires telecom and radio system (to enable communications with rest of SoS), the collection of ERUs
  -- and a directory of external organisations.
  public Insiel : TeleCom * RadioSystem * set of ERU * map CUS`ExtOrg to TeleCom`PhoneNumber ==> Insiel
Insiel(t, r,erus, extOrgs) ==
{
  phoneSys := new PhoneSystem(t, phoneLines);
  cus := new CUS(r, phoneSys, erus, cusRadCh, extOrgs);
  phoneSys.addCUS(cus);

  -- Add the CUS radio Id to each ERU's contact list
  for all e in set erus do
    {e.addContact(cusRadCh);
  }

  -- Operation to get the CUS of the Insiel object
  public getCUS: () ==> CUS
  getCUS() == return cus;
--operation to get the PhoneSystem of the Insiel object
public getPhoneSystem: () ==> PhoneSystem
getPhoneSystem() == return phoneSys;
end Insiel

PhoneSystem Class

-- Models the phone system as part of the Insiel call centre
-- Inherits the Communicator class
-- to enable a generic interface to the TeleCom
class PhoneSystem is subclass of Communicator

instance variables
-- Object references to allow communication with the TeleCom
-- and CUS
private telecom : TeleCom;
private cus : [CUS];

-- Queue of incoming calls
private queue : seq of TeleCom`CallId;
-- Calls currently in progress
private callsInProgress : set of TeleCom`CallId;
-- Recorded calls for auditing purposes
private recordedCalls : map TeleCom`CallId to seq of Message;

-- Invariant to ensure that all current calls are either in
-- the queue or are in progress and that the queue does not
-- contain duplicate calls
inv (len queue + card callsInProgress = card currentCalls) and
(forall c in set currentCalls & c in set elems queue
   or c in set callsInProgress) and
(elems queue inter callsInProgress = {}) and
(card elems queue = len queue);

operations

-- Constructor - initialise instance the TeleCom system
-- and the number of phone lines, plus the remaining
-- variables to empty sets/sequences/mappings
public PhoneSystem : TeleCom * nat==> PhoneSystem
PhoneSystem(t, pl) ==
{
    queue := [];
callsInProgress := {};
cus := nil;
    telecom := t;
    recordedCalls := {||};
    phoneLines := pl;
    currentCalls := {};
};

-- Add new call to the end of the queue
public initiateCall : TeleCom`CallId ==> ()
initiateCall(cid) ==
    queue := queue ^ [cid]
pre cid not in set elems queue;

-- Remove a call from the front of the queue, add it to the
-- set of calls in progress and start recording the call
public answerCall : () ==> TeleCom`CallId
answerCall() ==
{ dcl call : TeleCom`CallId := hd queue;
callsInProgress := callsInProgress union {call};
queue := tl queue;
initiateRecordCall(call);
return call }
pre not isEmptyQueue()
-- no calls are lost in this process
post elems queue union callsInProgress =
    elems queue~ union callsInProgress~;

-- place a call to a given phone number
public placeCall : TeleCom`PhoneNumber * PhoneSystem ==> bool
placeCall(d, s) ==
{ -- place a call and get a CallId
dcl call : [TeleCom`CallId] := telecom.placeCall(d, s);
-- if placing a call is successful...
if call <> nil then
    { -- place the call in calls in progress.
callsInProgress := callsInProgress union {call};
currentCalls := currentCalls union {call};
return true
}
else
    -- if placing a call is not successful...
return false
}
pre card currentCalls < phoneLines and
    telecom.isValidPhoneNumber(d);
--End a call that is currently in progress
public endCall : TeleCom\CallId ==> ()
endCall(call) ==
  (callsInProgress := callsInProgress \ \{call\};
currentCalls := currentCalls \ \{call\};
)
pre call in set callsInProgress and
  call in set currentCalls
post card callsInProgress = card callsInProgress~ - 1 and
  card currentCalls = card currentCalls~ - 1;

--Return all current calls in progress
public getCurrCalls : () ==> set of TeleCom\CallId
getCurrCalls() ==
  return callsInProgress;

--Start recording a new call. Places a new entry in the
--recorded calls variable
private initiateRecordCall : TeleCom\CallId ==> ()
initiateRecordCall(call) ==
  recordedCalls := recordedCalls union \{call|->[]\}
pre call not in set dom recordedCalls;

--Get recorded messages for a given call
private getRecordedCall : TeleCom\CallId ==>
-seq of Message
getRecordedCall(call) ==
  return recordedCalls(call)
pre call in set dom recordedCalls;

--Record a message as passed between participants of a
--given call
private recordMessage : TeleCom\CallId * Message ==>
() recordMessage(call, msg) ==
  recordedCalls := recordedCalls ++
    \{call |-> recordedCalls(call) ^ [msg]\}
pre call in set dom recordedCalls;

--Allows messages to be sent to the Caller (through the
--TeleCom) from the CUS. Return message may be nil if no
--response is given by the Caller
public sendMessage: TeleCom\CallId * Message ==>
[Message] sendMessage(cid, msg) ==
  (dcl returnedMessage : Message;
   recordMessage(cid, msg);
   returnedMessage := telecom.converse(cid, self, msg);
   recordMessage(cid, returnedMessage);
   return returnedMessage
)
pre cid in set callsInProgress;

--Responding to incoming messages from the TeleCom
public converse : TeleCom\CallId * Message ==> [Message]
converse(cid, msg) == is not yet specified
pre cus <> nil;

public divertToExternalOrg : TeleCom\CallId *
TeleCom\PhoneNumber ==> ()
divertToExternalOrg(c, p) == is not yet specified
pre telecom.isValidPhoneNumber(p);

public isEmptyQueue : () ==> bool
isEmptyQueue() ==
return queue = [];

public addCUS : CUS ==> ()
addCUS(c) ==
cus := c
pre cus = nil;

end PhoneSystem

RadioReceiver Class

--Abstract class to enable radio communications.
class RadioReceiver
types
    public RadioReceiverId = token;

instance variables
    protected rId : RadioReceiverId;
    protected radioSys : RadioSystem;

operations
    --RadioReceiver Constructor
    --Initialises its radio channel and the RadioSystem
    public RadioReceiver : RadioReceiverId * RadioSystem ==> RadioReceiver
    RadioReceiver(id, r) ==
    { 
        rId := id;
        radioSys := r;
    }

    --Placeholder operation to enable a RadioReceiver to
    --send a radio message to another RadioReceiver
    public sendMessage : RadioReceiverId * RadioReceiverId *
    RadioSystem\RadioMessage ==> [RadioSystem\RadioMessage]
sendMessage(s, r, msg) == is subclass responsibility
pre \(s = \text{rId};\)

--Placeholder operation to enable a RadioReceiver to
--receive a radio message from another RadioReceiver
public receiveMessage : RadioReceiverId * RadioReceiverId
    * RadioSystem`RadioMessage ==> ()
receiveMessage(s, r, msg) == is subclass responsibility
pre \(r = \text{rId};\)

end RadioReceiver

RadioSystem Class

--RadioSystem class representing the communication mechanism
--between the CUS and ERUs
class RadioSystem

types

    public RadioMessage = token;

instance variables

--Directory of radio channels
channels : map RadioReceiver`RadioReceiverId to RadioReceiver;
--invariant to ensure 1-to-1 mapping in channels
inv card dom channels = card rng channels;

operations

--RadioSystem Constructor
--Initialises the mapping from radio channels to radio
--receivers
public RadioSystem : map RadioReceiver`RadioReceiverId
to RadioReceiver ==> RadioSystem
RadioSystem(c) ==
{
    channels := c;
};

--Send a message from one radio receiver to another. Radio
--receivers must have channels which exist in the channels
--directory
public sendMessage: RadioReceiver`RadioReceiverId *
    RadioReceiver`RadioReceiverId * RadioMessage ==> ()
sendMessage(send, rec, msg) ==
{
    channels(rec).receiveMessage(send, rec, msg)
}  
pre \{send, rec\} subset dom channels;

--Add a collection of channels to the directory of channels
public addChannels: map RadioReceiver`RadioReceiverId to
    RadioReceiver ==> ()
addChannels(ch) ==
{
channels := channels \( \cup \) ch;
}

\textbf{pre} \text{ dom} \quad \text{channels} \quad \text{inter} \quad \text{dom} \quad ch = \{\} \quad \text{and} \quad \text{rng} \quad \text{channels} \quad \text{inter} \quad \text{rng} \quad ch = \{\};

-- Add a new radio channel to the directory of channels
-- The radio Id or radio receiver may not already exist in
-- the directory

\textbf{public} \quad \text{addChannel} : \quad \text{RadioReceiver}``RadioReceiverId * \quad \text{RadioReceiver} \Rightarrow ()

\text{addChannel}(\text{radioId}, \text{receiver}) ==
{
  \text{channels} := \text{channels} \cup \{\text{radioId} |\to \text{receiver}\};
}

\textbf{pre} \quad \text{radioId} \quad \text{not in} \quad \text{set} \quad \text{dom} \quad \text{channels} \quad \text{and} \quad \text{receiver} \quad \text{not in} \quad \text{set} \quad \text{rng} \quad \text{channels};

\textbf{end} \quad \text{RadioSystem}

\section*{SoS Class}

-- Class to contain all constituent systems of the emergency response
-- SoS

\textbf{class} \quad \text{SoS}

\textbf{instance variables}

insiel: \text{Insiel};

\text{telecom : TeleCom;}

erus : \text{set of} \text{ERU;}

\text{radio : RadioSystem;}

\textbf{values}

\text{insPN : TeleCom}``PhoneNumber = \text{mk_token}("118");

\textbf{operations}

-- SoS Constructor
-- Initialises the Insiel System, the TeleCom System, the
-- ERUs and the RadioSystem

\textbf{public} \quad \text{SoS} : \quad \text{map} \quad \text{CUS}``ExtOrg \text{ to} \text{TeleCom}``PhoneNumber \text{ *}
\quad \text{set of} \text{ERU} \text{ *} \quad \text{RadioSystem} \text{ *} \quad \text{TeleCom} \Rightarrow \text{SoS}

\text{SoS}(\text{extOrgs}, \text{es}, \text{rad}, \text{tc}) ==
{
  -- declare local PhoneSystem variable.
  \text{dcl} \quad \text{ps} : \text{PhoneSystem};

  \text{erus} := \text{es;}
  \text{radio} := \text{rad;}
  \text{telecom} := \text{tc;}
  \text{insiel} := \text{new} \text{Insiel(telecom, radio, erus, extOrgs);};

  -- Add Insiel’s phone system to the TeleCom directory
  \text{ps} := \text{insiel.getPhoneSystem();}
  \text{telecom.addCommunicator(\text{insPN,ps});}
}
--Returns the telecommunications system of the SoS
public getTeleCom : () ==> TeleCom
getTeleCom() == return telecom;

end SoS

TeleCom Class

--Class to model the TeleCom system - enabling communications
--between Communicator objects (Callers and PhoneSystem).
--Telecom is an externally managed system.
class TeleCom
types

public PhoneNumber = token;

public TelcoCall :: dialled : Communicator
source : Communicator
timestamp : token;

public CallId = token;

instance variables

--maps for current calls, call history and phones
currentCalls : map CallId to TelcoCall;
callHistory : map CallId to TelcoCall;
communicators : map PhoneNumber to Communicator;

--Invariant to ensure that:
-- 1. 1-to-1 mapping in communicators
-- 2. No call is both current and history
-- 3. Current calls is consistent with the communicators
-- 4. No calls in the history are thought to be current by
--   the communicators involved
inv card dom communicators = card rng communicators and
dom currentCalls inter dom callHistory = {} and
(forall c in set dom currentCalls &
   currentCalls(c).dialled.isCurrentCall(c) and
   currentCalls(c).source.isCurrentCall(c)) and
(forall c in set dom callHistory & not
   currentCalls(c).dialled.isCurrentCall(c) and not
   currentCalls(c).source.isCurrentCall(c));

operations

--TeleCom Constructor
--Initialises the mapping from phone numbers to
--communicators, the current calls and the calls history
public TeleCom : map PhoneNumber to Communicator ==> TeleCom
TeleCom(c) ==
{
  communicators := c;
currentCalls := {)->};
callHistory := {\langle \rangle}
);

--Places a call to a given phone number
--Requires that the communicators are known to the TeleCom
--and that the source has a free phone line
public placeCall: PhoneNumber * Communicator ==> [CallId]
placeCall(d, s) ==
  --if the destination has a free phone line...
  if communicators(d).freePhoneLine()
  then
    --Generate new call identifier & telco call record
    dcl call : CallId := generateNewID();
    currentCalls := currentCalls munion
      {call \langle \rangle mk_TelcoCall(communicators(d),
                  s, getTime())};
    --initiate the call for the source
    s.initiateCall(call);
    --initiate the call for the destination
    communicators(d).initiateCall(call);
    return call
  )
  --if the destination does not have a free phone line...
  else
    return nil
pre s in set rng communicators and
  d in set dom communicators and
  s.freePhoneLine();

--Sends a message from one communicator to the other and
--returns the given response.
--Requires that the call is current and the source is a
--participant in the call.
--Has been designed to be generic, but will only use it in
--one way for current case study
public converse : CallId * Communicator *
  Communicator\Message ==> [Communicator\Message]
converse(cid, src, msg) ==
  if currentCalls(cid).dialled = src
  then return currentCalls(cid).source.converse(cid,msg)
  else return currentCalls(cid).dialled.converse(cid,msg)
pre cid in set dom currentCalls and
  (currentCalls(cid).dialled = src or
  currentCalls(cid).source = src);

--Ends a call for a given CallId and place the call in the
--call history.
--Requires that the call is current and that the request to
--end the call comes from one of the participants of the
--call.
public endCall : CallId * Communicator ==> ()
endCall(cid, com) ==
{ currentCalls(cid).source.endCall(cid);
  currentCalls(cid).dialled.endCall(cid);
  callHistory := callHistory ++ {cid \langle \rangle currentCalls(cid)};
  currentCalls := {cid} <-: currentCalls;
\[
\begin{align*}
\text{pre} & \text{ cid in set dom currentCalls and} \\
& \quad \text{(currentCalls(cid).dialled = com or} \\
& \quad \quad \text{currentCalls(cid).source = com) and} \\
& \quad \text{cid not in set dom callHistory} \\
\text{post} & \quad \text{cid not in set dom currentCalls and} \\
& \quad \text{cid in set dom callHistory;} \\
\end{align*}
\]

```
--Add a collection of communicators to the directory of phone numbers
public addCommunicators: map PhoneNumber to Communicator ==> ()
addCommunicators(comms) ==
  (communicators := communicators munion comms;)
pre dom communicators inter dom comms = {} and
  rng communicators inter rng comms = {};
```

```
--Add a communicator to the directory of phone numbers
public addCommunicator: PhoneNumber * Communicator ==> ()
addCommunicator(pn, comm) ==
  (communicators := communicators munion {pn |-> comm};)
pre pn not in set dom communicators and
  comm not in set rng communicators;
```

```
--Operation to define a new CallId - implicitly defined
private generateNewID() id : CallId
ext rd currentCalls
  rd callHistory
post id not in set (dom currentCalls union dom callHistory);
```

```
--Returns true when given a valid phone number
public isValidPhoneNumber : PhoneNumber ==> bool
isValidPhoneNumber(p) ==
  return p in set dom communicators;
```

```
--Returns the current time
private getTime: () -> token
getime() == is not yet specified;
```

end TeleCom

### TestClass Class

```
--Class to create test objects
class TestClass
values
  extOrgs : map CUS`ExtOrg to TeleCom`PhoneNumber =
    (<Fire> |-> mk_token("115"),
```
<Police> |-> mk_token("113")
<ForestRanger> |-> mk_token("123")
<CivilProtection> |-> mk_token("432")

instance variables

-- empty telecom, populated in constructor
tc : TeleCom := new TeleCom({});

-- empty radio system, populated in constructor
radio : RadioSystem := new RadioSystem({});

-- illustrative set of callers
c1 : Caller := new Caller(tc);
c2 : Caller := new Caller(tc);
c3 : Caller := new Caller(tc);
c4 : Caller := new Caller(tc);
c5 : Caller := new Caller(tc);

callers : set of Caller := {c1, c2, c3, c4, c5};

-- illustrative telecommunications mapping
communicators : map TeleCom`PhoneNumber to Communicator :=
{
    mk_token("0432750783") |-> c1,
    mk_token("0432739809") |-> c2,
    mk_token("0432768201") |-> c3,
    mk_token("0432742819") |-> c4,
    mk_token("0432750294") |-> c5,
    mk_token("115") |-> new Communicator(2),
    mk_token("113") |-> new Communicator(4),
    mk_token("123") |-> new Communicator(5),
    mk_token("432") |-> new Communicator(2)
};

-- illustrative set of radio identifiers
e1radId : RadioReceiver`RadioReceiverId := mk_token("ERU1rad");
e2radId : RadioReceiver`RadioReceiverId := mk_token("ERU2rad");
e3radId : RadioReceiver`RadioReceiverId := mk_token("ERU3rad");

-- illustrative set of ERUs
e1 : ERU := new ERU(mk_token("ERU1"), e1radId, radio,
    {e2radId, e3radId});
e2 : ERU := new ERU(mk_token("ERU2"), e2radId, radio,
    {e1radId, e3radId});
e3 : ERU := new ERU(mk_token("ERU3"), e3radId, radio,
    {e1radId, e2radId});

erus : set of ERU := {e1, e2, e3};

-- illustrative radio channels mapping
chanMap : map RadioReceiver`RadioReceiverId to RadioReceiver :=
{
    e1radId |-> e1.getRadioReceiver(),
    e2radId |-> e2.getRadioReceiver(),
    e3radId |-> e3.getRadioReceiver()
};
env : Environment;
sos : SoS;

--Invariant to ensure that the ERU identifiers are a subset
--of chanMap and the mapping is consistent with the ids
--stored by the ERUs
inv forall e in set erus &
    e.getRadioId() in set dom chanMap and
    e.getRadioReceiver() = chanMap(e.getRadioId());

operations

--TestClass Constructor
--Initialises the SoS and its environment
--Adds the radio channels mapping to the RadioSystem and
--the telecommunications mapping to the PhoneSystem
public TestClass : () ==> TestClass
TestClass() ==
{
    env := new Environment(callers);
    radio.addChannels(chanMap);
    tc.addCommunicators(communicators);
    sos := new SoS(extOrgs, erus, radio, tc)
}

--main operation used for testing the model
public main : () ==> ()
main() ==
{
    -- exercise the model for testing purposes
    dcl test : TestClass := new TestClass();
    return;
}

end TestClass
Annex II

8 Model overview

8.1 [Package] baseline SoS Overview

Description: The baseline Insiel SoS configuration takes into account four systems:

- The "CUS" (Centrale Unica di Soccorso = unified rescue central, Unified Emergency Call Centre) is the core system in the ERS SoS
- The ERUs (Emergency Response Units, usually consisting of an ambulance or flycar and a crew such as a driver and medical staff)
- A Radio System for the communication among the CUS and the ERUs
- A Phone System for the communication to the CUS of incoming casualty events as well as for systems which are outside the boundary of the case study (Hospitals, External Organizations such as Police, Fire Brigades, Civil Protection and so on)

The SoS constituent systems are expected to follow a predicted operational protocol during the delivery of services, however fault conditions may occur. In such cases, based on the mission criticality, different redundant technologies may be used to recover the normal operation flow; some of them are considered in the model.

The Phone System and the Radio System, from a behavioural point of view, can be considered simple interfaces respectively between the telephony provider (Telecom) system and the CUS as well as between the ERUs and the CUS. However, they are considered as systems because they are made up of third party components, they are relevant for the fulfilment of non-functional requirements (such as performance) and they are critical for possible failure conditions.

Theoretically, the Radio System is independent, however it can share some resources (such as conversation recording devices), for management and budget reasons.
8.2 [Package] improved SoS Overview

Figure 24 [Package] improved SoS Overview
Type: BlockDefinitionDiagram
Description: SoS overview Structure- A Mobile Phone System has been added to provide the capability of resuming faults of the Radio System

9 Requirements

9.1 Functional Requirements
### Table 3 Functional Requirements

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>req_00_01</td>
<td>provide statistics. The CUS shall provide statistics with special attention to the time from the emergency call to the ERU intervention and to the delivery in hospital.</td>
</tr>
<tr>
<td>req_Auth01</td>
<td>define operator roles. The System shall allow the configuration of the operations permitted to each operator or role of an operator. An operator can change role.</td>
</tr>
<tr>
<td>req_Auth02</td>
<td>definition of user profiles and assignment of roles to users. When a user is authenticated, the System shall enable operations on the basis of the roles the user can embody.</td>
</tr>
<tr>
<td>req_Auth03</td>
<td>trace operator/role activity. The System shall trace the operations performed by operators, with particular attention to accessing sensitive data of patients and other people.</td>
</tr>
<tr>
<td>req_Auth04</td>
<td>retrieve settings and working preferences of operator/role. When a user is authenticated in the system, if the user is a CUS operator, the workstation user interface will be set up with personal preferences including frequent calls directory and headset volume</td>
</tr>
<tr>
<td>req_CUS_000000</td>
<td>For each incoming call, if appropriate, activate the emergency response (general requisite, detailed by sub-requisites). The CUS system (with intervention of the operator) assesses the response needs and activates the required resources</td>
</tr>
<tr>
<td>req_CUS_000100</td>
<td>mapping facility. This includes browsing maps and other features that do not need detailing at the SoS level</td>
</tr>
<tr>
<td>req_CUS_010000</td>
<td>manage and save call information (detailed by sub-requisites).</td>
</tr>
<tr>
<td>req_CUS_0101</td>
<td>create a 'record' for call information and save it with the type of action flag when completed. Caller's phone number is mandatory. see &quot;Call info bag&quot; in model</td>
</tr>
<tr>
<td>req_CUS_0102</td>
<td>The 'record' containing call information ( &quot;Call info bag&quot; ) will include the result of the operation. Not pertinent calls (wrong number and so on) and calls diverted to other organizations shall be logged with a proper result code.</td>
</tr>
<tr>
<td>req_CUS_0103</td>
<td>The 'record' containing call information ( &quot;Call info bag&quot; ), besides result of operation, will contain casualty details whenever applicable.</td>
</tr>
<tr>
<td>req_CUS_0105</td>
<td>provide a User Interface for recording the information provided by the caller. The CUS software shall enable the operator filling the Call info bag</td>
</tr>
<tr>
<td>req_CUS_0106</td>
<td>associate the call with the response events it gives rise to. The &quot;Call info bag&quot; logged after a call should be an entry point to access response and mission data.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>req_CUS_0107</td>
<td>display recent calls information. Incoming emergency calls shall be displayed in the user interface of the CUS software with the caller’s phone number and other relevant information. This is a different requirement from &quot;req_Phone0101&quot; as here the focus is on the incoming calls and having them attached to information.</td>
</tr>
<tr>
<td>req_CUS_0108</td>
<td>retrieve previous call information through the appropriate keys. The System shall enable searching the Call info bag with keys such as the caller’s phone number, the target location address, the caller name</td>
</tr>
<tr>
<td>req_CUS_02000000</td>
<td>If a response is needed, send one or more ERU with the correct equipment to the identified target (general requisite, detailed by sub-requisites). The CUS shall identify the needed equipment, verify the availability of ERUs with such equipment in idle state of on missions with lower priority, and assign them the mission immediately</td>
</tr>
<tr>
<td>req_CUS_02010000</td>
<td>If needed, pre-alert the hospital for admission. When a response or rescue needs hospitalization of one or more patients, it is necessary to verify the availability of Hospitals for admission and to prepare possible surgery or other intervention.</td>
</tr>
<tr>
<td>req_CUS_02010100</td>
<td>identify Patient.</td>
</tr>
<tr>
<td>req_CUS_02010101</td>
<td>identify Patient needs.</td>
</tr>
<tr>
<td>req_CUS_02010102</td>
<td>The System shall interface with the Regional health registry</td>
</tr>
<tr>
<td>req_CUS_020200</td>
<td>Identify timing needs for equipment or admission to hospital. Assess priority of response actions. This is accomplished by CUS operators.</td>
</tr>
<tr>
<td>req_CUS_020201</td>
<td>Identify need of flycar before ambulance</td>
</tr>
<tr>
<td>req_CUS_020202</td>
<td>Identify need for helicopter</td>
</tr>
<tr>
<td>req_CUS_020203</td>
<td>Identify need for diversion of vehicle already in proximity of target</td>
</tr>
<tr>
<td>req_CUS_020300</td>
<td>Identify target location. The CUS software shall assist operator in the identification of target location on the basis of the description provided by the caller.</td>
</tr>
<tr>
<td>req_CUS_020400</td>
<td>Identify equipment needs (call centre)</td>
</tr>
<tr>
<td>req_CUS_020401</td>
<td>Identify need for external organization</td>
</tr>
<tr>
<td>req_CUS_020402</td>
<td>Identify need for admission to a particular hospital</td>
</tr>
<tr>
<td>req_CUS_020600</td>
<td>Transmit target data to suitable ERU</td>
</tr>
<tr>
<td>req_CUS_030100</td>
<td>Maintain data about available equipment on ERUs</td>
</tr>
<tr>
<td>req_CUS_030200</td>
<td>Maintain data about vehicle availability. The CUS shall keep</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>updated information on the status of the ERU vehicles as available or on mission.</td>
<td></td>
</tr>
<tr>
<td>req_CUS_030300</td>
<td>Maintain data about vehicle position. The CUS shall keep updated information on the ERU geographical position: baseline approximate information are the mission starting point and target. Improvements include continuous tracking in the map provider</td>
</tr>
<tr>
<td>req_Phone00</td>
<td>The System shall handle phone calls (general requisite, detailed by sub-requisites)</td>
</tr>
<tr>
<td>req_Phone0100</td>
<td>make outgoing call. CUS operators should be enabled to perform outgoing calls, in order to alert external organizations, recall the number from which the emergency was signalled and other purposes.</td>
</tr>
<tr>
<td>req_Phone0101</td>
<td>retrieve the last dialled numbers. The System shall keep trace of recently dialled number for quick redialling</td>
</tr>
<tr>
<td>req_Phone0102</td>
<td>retrieve a number from a user-configured directory. The System shall allow the user to maintain a directory of frequently dialled numbers, in order to speed-up dialling.</td>
</tr>
<tr>
<td>req_Phone0103</td>
<td>dial again last called number</td>
</tr>
<tr>
<td>req_Phone0104</td>
<td>dial a generic number, not present in recent call list or user directory</td>
</tr>
<tr>
<td>req_Phone0200</td>
<td>enqueue and handle incoming calls. The System shall add calls in the appropriate queues on the basis of incoming phone lines. Queues will be accessible by the CUS system in order to display them in a user interface available to operators. Selecting a call from the queue and answering it removes the call from queue.</td>
</tr>
<tr>
<td>req_Phone0201</td>
<td>display incoming calls in queues according to incoming phone line. The System shall visualize incoming calls into different queues assigned to different priority. Calls from the lines answering the emergency number 118 will be added to the emergency queue that has the highest priority. Calls which are in hold on/standby status are moved to a lower priority queue. A queue is dedicated to calls coming from external organizations which contribute to casualty responses. Lowest priority is given to green numbers and calls from external organizations not contributing to casualty responses.</td>
</tr>
<tr>
<td>req_Phone0300</td>
<td>hold on/standby current conversation. The System shall enable the operator to standby a call in progress: the standby call will be moved to the proper queue in order to make it visible to the operator user interface of the CUS system.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>req_Phone0400</td>
<td>divert call. The operators should be enabled to divert calls that must be addressed by other organizations diversion</td>
</tr>
<tr>
<td>req_Phone0500</td>
<td>visualize the id of the line hosting the conversation. The System shall highlight the phone line on the operator user interface of the CUS system.</td>
</tr>
<tr>
<td>req_Phone0600</td>
<td>record each phone conversation and provide tools to listen to recordings. The System shall record all conversations for security (hearing again in case of difficult understanding) or legal (verification of responsibilities) reasons.</td>
</tr>
<tr>
<td>req_Phone0700</td>
<td>Allow other calls to be processed in standby. The System shall enable the operator to standby a call in progress, in order to pick-up and process other new calls or higher-priority calls.</td>
</tr>
<tr>
<td>req_Phone0800</td>
<td>setup phone appliance with user configuration. The System shall recognize the operator authentication from the CUS system software and setup the phone with personal preferences (e.g. headphones, microphone and ringtone volume).</td>
</tr>
<tr>
<td>req_Phone0900</td>
<td>visualize status of phone appliance and of its connection. The Phone System and computer telephony parts in the CUS Systems shall auto-diagnose and highlight problems in computer telephony, so that the operators can switch to using emergency telephone appliances.</td>
</tr>
<tr>
<td>req_Phone0901</td>
<td>highlight phone appliance anomalies</td>
</tr>
<tr>
<td>req_radio_0100</td>
<td>make and receive radio call. The System shall allow radio communication of CUS operators with the ERU crew.</td>
</tr>
<tr>
<td>req_radio_0101</td>
<td>select radio channel from directory of ERUs and external organizations</td>
</tr>
<tr>
<td>req_radio_0200</td>
<td>enqueue and handle incoming radio calls. The System shall encode possible radio calls from different ERUs in a similar way to phone calls</td>
</tr>
<tr>
<td>req_radio_0300</td>
<td>send and receive radio message. The System shall allow communication of status messages in text form</td>
</tr>
<tr>
<td>req_radio_0400</td>
<td>send and receive data via radio. This is an improvement: ERUs and CUS should exchange geographical coordinates in a suitable form to help the navigation of the ERU driver or to update the ERU position on the map. Similarly, they should be able to exchange other types of data such as recordings from health instruments.</td>
</tr>
<tr>
<td>req_radio_0500</td>
<td>record each radio conversation and provide tools to listen to recordings. The System shall record all conversations for security (hearing again in case of difficult understanding) or legal (verification of responsibilities) reasons</td>
</tr>
</tbody>
</table>
9.2 Non Functional Requirements

![Diagram of Non Functional Requirements]

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reqnf_f01</td>
<td>enable other call centre to perform operation for the call centre not working. In case of severe fault of more parts of a CUS central, another central may handle the processes on its behalf. The deputed CUS will access a copy of the data concerning emergency responses and missions which is mirrored over the regional network. Emergency call directed to the CUS out of service should be redirected from telecoms to the deputy CUS.</td>
</tr>
<tr>
<td>reqnf_f02</td>
<td>provide continuous connectivity with ERUs. If ERUs are in areas with shaded radio signal, other means must be adopted to provide connectivity with the CUS</td>
</tr>
<tr>
<td>reqnf_fault</td>
<td>fault tolerance. The System shall allow alternative technologies or procedures to operate in case of faults in the CUS or related systems</td>
</tr>
</tbody>
</table>

10 SoS UCM - Sos overview

Before getting into structure details, we highlight the most important use cases from our viewpoint (the viewpoint of the CUS).
10.1 Process Emergency (CUS viewpoint)

**Figure 25 Process Emergency (CUS viewpoint)**

Type: Use Case Diagram

Description: The above diagram represents the macroscopic behaviours exposed by the SoS that are mainly invoked by the Call Centre operator who is responsible for:

- Receiving the emergency call and collecting all necessary information on the accident
- Eventually alerting the most appropriate health structure depending on the location and the nature of the accident
- Coordinating the ERUs attending the response missions
- Monitoring response and resources

11 Block Diagrams

11.1 Block Definition Diagrams

Two simplified block definition diagrams are reported in section 8 (Structure Overview). Here the other diagrams are collected.
11.1.1 SoS overview Structure-baseline

![SoS overview Structure-baseline diagram](image)

**Figure 26 SoS overview Structure-baseline**

**Type: BlockDefinitionDiagram**

Description: This diagram shows the systems with a focus on their relation to the case study SoS. For a more detailed description see description of diagram [Package baseline SoS Overview CUS] is the only completely custom system in the SoS.

11.1.2 SoS overview Structure- Fault Tolerant

![SoS overview Structure-Fault Tolerant diagram](image)

**Figure 27 SoS overview Structure-Fault Tolerant**

**Type: BlockDefinitionDiagram**

Description: This diagram shows the changes made to the diagram SoS overview Structure-baseline in order to add a Mobile Phone System. This addition is intended to provide fault tolerance for the communication of CUS with ERUs and has been considered in the case study as a proof of concept of modelling the addition of a new system to the SoS.
11.1.3 Insiel SoS key entity structure

![Insiel SoS key entity structure diagram](image)

**Figure 28 Insiel SoS key entity structure**

**Type: BlockDefinitionDiagram**

Description: This diagram collects the keywords used in the analysis made for an early version of the CUS software.

The *Incoming Call* is further investigated in the diagram *Incoming Call keyword usage*.

A part of the *Incoming Call* (see *Call info bag*, see also * Incoming Call keyword usage* diagram), *Emergency Response* and *Mission* correspond to behavioural actions but (for managing *CUS Scheduling* and *CUS Monitoring*) they are implemented as software classes as well, because behavioural actions need tracing. In a future development, *Emergency Response* and *Mission* might deserve further detailing, in a similar way to what has been done for the *Incoming Call*. 

---

123
11.1.4 Incoming Call keyword usage

![Diagram of Incoming Call keyword usage](image)

**Figure 29 Incoming Call keyword usage**

**Type: BlockDefinitionDiagram**

Description: The term "call" refers to events on the phone lines (Phone call), and vocal (audio) conversation.

A call to the emergency number 118 (Incoming Call or Emergency call) requires carrying out the conversation following a best practice protocol and compiling compilation the logs (Call info bag) of the interview results and actions taken. These logs can be updated with possible subsequent information about the Emergency Response triggered by the call (see also Insiel SoS key entity structure diagram).
11.1.5 Insiel behavioural blocks structure diagram

![Diagram showing Insiel behavioural blocks structure diagram](image)

**Figure 30** Insiel behavioural blocks structure diagram

**Type:** BlockDefinitionDiagram

**Description:** This diagram details the SoS overview Structure-baseline, showing how the CUS accomplishes different functionalities that can in principle be fulfilled by different subsystems.
11.1.6 Insiel SoS deployment structure

Figure 31 Insiel SoS deployment structure

Type: BlockDefinitionDiagram

Description: This diagram details the SoS overview Structure-baseline, with focus on the physical parts of systems, where specific functionalities and ports may reside. This can be a draft for a future UML deployment diagram.
11.1.7 Call Centre "CUS" internal Structure

Description: The main challenge in modelling the System was understanding the different roles of CUS as these functionalities can be represented by distinct behavioural interfaces but had been analysed as a whole.

Such functionalities are:

1) **CUS Phone Answering**.
2) **CUS Scheduling**.
3) **CUS Monitoring**.

Whilst both **CUS Scheduling** and **CUS Monitoring** expose communication interfaces with the **Radio System** and need functionalities for outgoing phone calls, **CUS Phone Answering** communicates with the **Phone System** only, using interfaces for handling the phone queue, answering the phone and diverting the call.
The CUS system, in spite of its complexity, is still considered as a whole because of parts shared by all three functionalities (such as the Mapping support, see Map provider), and because of the strictly interdependent communication interfaces among the three, which at the moment result in using a unique software package. Also other ERS on the market combine the management of phone answering and mission scheduling and monitoring within a unique management software.

Three different internal block diagrams were used to focus on the interfaces used by the three functionalities above. The CUS Monitoring internal block diagram is very similar to CUS Scheduling and shares the same communication needs with the ERUs.

### 11.1.8 Case Study Processes

![Diagram showing Case Study Processes](Figure 33 Case Study Processes)

**Type:** BlockDefinitionDiagram
Description: This diagram highlights the processes investigated with behavioural diagrams such as activity diagrams

11.2 Internal Block Diagrams

11.2.1 Case Study IBD

![Case Study IBD Diagram](image)

Figure 34 Case Study IBD

Type: InternalBlockDiagram

Description: this diagram highlights the main interfaces of interest for the investigation of the Radio System failure (shaded radio signal or errors of greater severity) using the baseline SoS (without mobile phone system).

These interfaces are described in Table 5.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIF acquire Datcom data</td>
<td>improvement of the baseline system: the interface is exposed by the CUS server to the Radio System to support the reception of binary data transmitted via radio</td>
</tr>
<tr>
<td>IIF CT Phone handling 0</td>
<td>interface exposed by Phone system to CUS</td>
</tr>
<tr>
<td>IIF Radio Select channel</td>
<td>interface provided by Radio System to set the radio channel (associated to ERU) to be used by CUS human operator for conversation</td>
</tr>
<tr>
<td>IIF RadioMsg</td>
<td>send encoded text data via radio signal</td>
</tr>
<tr>
<td>IIF RadioVoice</td>
<td>send vocal communication modulated on radio signal (could also be encrypted in digital radio communication). Traditional voice technology allows only one speaker to transmit on the same channel at a time so that CUS/ERU communication needs an &quot;_over&quot; signal to tell the other operator that the channel is free for responding.</td>
</tr>
<tr>
<td>IIF send data</td>
<td>improvement to baseline system: the interface is exposed by the Radio</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>to datcom</td>
<td>System to the CUS server to send binary data via radio</td>
</tr>
<tr>
<td>IIF voice</td>
<td>passing voice via cable, this interface is used for computer telephony but also for connecting the operator workstation of the CUS to the Radio System. For the latter, the &quot;_over&quot; message is included.</td>
</tr>
</tbody>
</table>

### 11.2.2 [block] Radio System

Figure 35 [block] Radio System

**Type: InternalBlockDiagram**

Description: At the SoS level of the case study, the Radio System is modelled as black boxes and only interfaces have been considered.
11.2.3 [block] Phone System

**Table 6 Interfaces of the Phone System**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIF CT Phone call</td>
<td>events triggered by the Phone System to automate some tasks which the CUS human operator would perform on the IIF call UI</td>
<td>Interface</td>
</tr>
<tr>
<td>IIF CT Phone handling 0</td>
<td>interface exposed by the Phone system to the CUS</td>
<td>Interface</td>
</tr>
<tr>
<td>IIF EnqueueCall</td>
<td>interface within the Phone System ; a different interface is exposed to the CUS in order to access the</td>
<td>Interface</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Type</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>IIF Telecom In</td>
<td>interface exposed by Telecom companies for incoming calls</td>
<td>Interface</td>
</tr>
<tr>
<td>IIF Telecom out</td>
<td>interface exposed by Telecom companies for making calls</td>
<td>Interface</td>
</tr>
<tr>
<td>IIF use conversation recordings</td>
<td>interface provided by the recording part of the phone system, allowing access to recordings of conversations</td>
<td>Interface</td>
</tr>
</tbody>
</table>

11.2.4 [block] CUS external interfaces

![CUS external interfaces diagram]

**Figure 37 [block] CUS external interfaces**

**Type: InternalBlockDiagram**

Description: This diagram is a synthesis of the internal block diagrams [block] CUS Phone Answering, [block] CUS Scheduling and [block] CUS Monitoring we required in the SysML model to display the interfaces attached to the ports of the CUS system, so that their operations could be referenced in sequence diagrams without the need for creating specific timelines for the internal parts.
(The three mentioned diagrams [block] CUS Phone Answering, [block] CUS Scheduling and [block] CUS Monitoring attach interfaces to ports of internal parts instead of representing them as attached to ports of the system, so that they are not sufficient to have operations of interfaces accessible to sequence diagram timelines representing the CUS in its entirety)

Table 7 CUS external interfaces

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIF acquire Datcom data</td>
<td>improvement to the baseline system: the interface is exposed by the CUS server to the Radio System to receive the binary data transmitted via radio</td>
</tr>
<tr>
<td>IIF Call logging and management</td>
<td>interface exposed by the CUS server software to clients to allow the operators handling the call detail data</td>
</tr>
<tr>
<td>IIF CT Phone call events</td>
<td>events triggered by the Phone System to automate some tasks performed by the CUS operator on the <strong>IIF call UI</strong></td>
</tr>
<tr>
<td>IIF CT Phone conversation handling</td>
<td>interface exposed by the Phone system to the CUS clients / operators</td>
</tr>
<tr>
<td>IIF CT Phone queue handling</td>
<td>interface exposed by the Phone system to the CUS : verify if used by CUS clients or server</td>
</tr>
<tr>
<td>IIF CT Phone queue info</td>
<td>interface exposed by the Phone system to the CUS server</td>
</tr>
<tr>
<td>IIF Mobile Phone</td>
<td>interface for the mobile phone, including voice communication, messaging and data exchange.</td>
</tr>
<tr>
<td>IIF Radio Select channel</td>
<td>interface provided by the Radio System to set the radio channel (associated to ERU) to be used by CUS operator for communication</td>
</tr>
<tr>
<td>IIF send data to datcom</td>
<td>improvement to baseline system: exposed by the Radio System to the CUS server for sending binary data via radio</td>
</tr>
<tr>
<td>IIF txtMessage</td>
<td>interface for passing text message via cable, connecting the CUS operator to the Radio System.</td>
</tr>
<tr>
<td>IIF conversation use</td>
<td>interface provided by the recording part of the phone system, providing access to recordings of conversations</td>
</tr>
<tr>
<td>IIF voice</td>
<td>passing voice via cable, this interface is used for computer telephony but also for connecting the CUS operator to the Radio System. For the latter, the ”_over” message is included.</td>
</tr>
</tbody>
</table>
11.2.5 [block] CUS Phone Answering

Description: A human operator in the role of CallCentreOperator operates a phone and a user interface running on one or more pc ("CUS Client").

The pc (‘client’) of the operator workstation interacts with:

- A server hosting the management software in charge of saving information and the management software where information about the call is saved (see Call info bag). In the current implementation they are a web and database server interfaced with the client through an intranet IP protocol.

- The phone system, connecting with an audio device (typically headset) for computer telephony and software interfaces to handle call queue (see call queue) and the current conversation occurring through the audio device.

- The Map provider, hosting a GIS (Geographical Information System) to visualize on a map the correct location of the mission Target on map, before making it available to CUS Scheduling (see Locate position activity diagram).
More interfaces shown in this diagram include:

- Telephone sets, to bypass computer telephony in case of failure
- Connection of the software managing the information about calls with:
  - the part of the phone system handling computer telephony (CTI Server), in order to receive signals for the creation of call information logs. If these signals are not handled automatically, the operator can action them.
  - the part of the phone system in charge of recording the conversations, in order to provide lookup for recordings.

The interfaces are described in Table 8.

### Table 8 Interfaces of the CUS Phone Answering

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIF Call logging and management</td>
<td>interface exposed by the CUS server software to clients for the management of call data by operators</td>
</tr>
<tr>
<td>IIF call UI</td>
<td>user interface exposed by the CUS client to operators for the management of call data</td>
</tr>
<tr>
<td>IIF CT Phone call events</td>
<td>events triggered by the Phone System to automate some tasks performed by the CUS operator on the IIF call UI</td>
</tr>
<tr>
<td>IIF CT Phone conversation handling</td>
<td>interface exposed by the Phone system to CUS clients / human operators</td>
</tr>
<tr>
<td>IIF CT Phone queue handling</td>
<td>interface exposed by the Phone system to the CUS : verify if used by CUS clients or server</td>
</tr>
<tr>
<td>IIF CT Phone queue info</td>
<td>interface exposed by the Phone system to the CUS server</td>
</tr>
<tr>
<td>IIF GIS functions</td>
<td>interface of the GIS Mapping support to localize mission targets and position of ERUs</td>
</tr>
<tr>
<td>IIF use conversation recordings</td>
<td>interface provided by the recording part of the phone system to access the recordings of conversations</td>
</tr>
<tr>
<td>IIF voice</td>
<td>passing voice via cable, this interface is used for computer telephony but also for connecting the operator workstation of the CUS to the Radio System. For the latter, the &quot;_over&quot; message is included.</td>
</tr>
</tbody>
</table>
11.2.6 [block] CUS Scheduling

Description: This diagram represents the final revision of the CUS scheduling with the inclusion of peer-to-peer communication of the Radio System with the CUS server and the Mobile Phone System introduced to provide fault-proof behaviour.

The diagram displays a direct link from the Radio System to the CUS server (IIF acquire Datcom data / IIF send data to datcom), which represents an improvement compared to the actual technology, and can be regarded as the insertion of a new peer-to-peer communication among parts.

The interface IIF RadioMsg is vendor-dependent, while interfaces (IIF acquire Datcom data and IIF send data to datcom) can be made stable with a proper interface within the Radio System, including a proper component to commute the data obtained via the IIF RadioMsg protocol.

Figure 39 [block] CUS Scheduling

Type: InternalBlockDiagram
Adding a Mobile Phone System has been represented as a black-box exposing a unique interface for voice and data (IIF Mobile Phone), as mobile phone protocols are consolidated and the Telecom operator is completely shaded from the users’ point of view of the communication.

The interfaces IIF response event handling and IIF response UI enable the operator to attach the new response instance to the Call info bag containing the data of the emergency which gave rise to it (using the management software hosted on server). The Missions to be scheduled (not represented here as considered as a process and/or as a software class accessed via the IIF response event handling interface) involve the verification and confirmation of the ERU availability. Currently, the Human Operator manually sends and receives status messages to/from the ERUs (textual or voice communications) via the Radio System.

The Map provider hosts a GIS and provides the functionality to visualize the position of available or divertable ERUs and mission Target on the map, as well as to estimate the optimal mission route and routing times.

The interfaces are described in Table 9.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIF acquire Datcom data</td>
<td>improvement to baseline system: exposed by CUS server to Radio System to support reception of binary data transmitted via radio</td>
</tr>
<tr>
<td>IIF GIS functions</td>
<td>interface of GIS to localize mission targets and position of ERUs</td>
</tr>
<tr>
<td>IIF Mobile Phone</td>
<td>interface for the mobile phone: the mobile phone system is modelled as a black box and a single interface includes voice communication, messaging and data exchange.</td>
</tr>
<tr>
<td>IIF Radio Select channel</td>
<td>interface provided by the Radio System to set the radio channel</td>
</tr>
<tr>
<td></td>
<td>(associated to ERU) to be used by CUS human operator for conversation</td>
</tr>
<tr>
<td>IIF RadioMsg</td>
<td>send encoded text data via radio signal</td>
</tr>
<tr>
<td>IIF RadioVoice</td>
<td>send vocal communication modulated on radio signal (could also be encrypted in digital radio communication). Traditional voice technology allows only one speaker to transmit on the same channel at a time so that CUS/ERU communication needs an &quot;over&quot; signal to tell the other operator that the channel is free for responding.</td>
</tr>
<tr>
<td>IIF response event handling</td>
<td>interface exposed by the CUS server software to the CUS client providing operations to schedule response</td>
</tr>
<tr>
<td>IIF response UI</td>
<td>User Interface exposed by the CUS client to human operator providing operations to schedule response</td>
</tr>
<tr>
<td>IIF send data to datcom</td>
<td>improvement to the baseline system: this interface is exposed by the Radio System to the CUS server to support sending binary data to</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IIF voice</td>
<td>passing voice via cable, this interface is used for computer telephony but also for connecting the operator workstation of the CUS to the Radio System. For the latter, the &quot;_over&quot; message is included.</td>
</tr>
</tbody>
</table>

**11.2.7 [block] CUS Monitoring**

**Description:**
The interfaces IIF mission handling and IIF Mission UI enable the operator to update Mission and data of missions on the management software hosted on server. At the time being, the Human Operator receives status messages (textual or voice...
communications) from ERUs via the Radio System and manually updates status and position of ERUs.

Sending messages via the Radio System may be dependent on the vendor specific technology of radio devices installed on board or ERU vehicles, so that the corresponding interface is kept distinct from traditional radio vocal communication (differs from phone communication for the need of the ‘over’ signal). Map provider, hosts a GIS to visualize ERUs and mission Target on map.

Introduced in a second step, after the baseline model: this diagram represents the final revision, including peer-to-peer communication of radio system with the CUS server as well as the Mobile Phone System introduced to provide fault-proof behaviour when radio signal is shaded or radio communication undergoes other types of failure.

The diagram displays a direct link from radio system to CUS server (IIF acquire Datcom data / IIF send data to datcom), which represents an improvement compared to actual technology, and can be regarded as the insertion of a new peer-to-peer communication among parts.

The vendor-dependent interface is IIF RadioMsg, while interfaces (IIF acquire Datcom data and IIF send data to datcom) could be made stable by realization of a proper façade within the Radio System, including a proper component to commute data got via the “IIF RadioMsg protocol.

Adding a Mobile Phone System has been represented as a black-box exposing a unique interface for voice and data (IIF Mobile Phone), as mobile phone protocols are consolidated and the Telecom operator is completely shaded from the users’ point of view of the communication.

The interfaces are described in Table 10.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIF acquire Datcom data</td>
<td>improvement to baseline system: exposed by CUS server to Radio System to support reception of binary data transmitted via radio</td>
</tr>
<tr>
<td>IIF GIS functions</td>
<td>interface of GIS to localize mission targets and position of ERUs</td>
</tr>
<tr>
<td>IIF mission handling</td>
<td>interface exposed by CUS server software to CUS client providing operations to monitor missions</td>
</tr>
<tr>
<td>IIF Mission UI</td>
<td>User Interface exposed by CUS client to human operator providing operations to monitor missions</td>
</tr>
<tr>
<td>IIF Mobile Phone</td>
<td>interface for the mobile phone: the mobile phone system is modelled as a black box and a single interface includes voice communication, messaging and data exchange.</td>
</tr>
<tr>
<td>IIF Radio Select channel</td>
<td>interface provided by Radio System to set the radio channel (associated to ERU) to be used by CUS human operator for communication</td>
</tr>
<tr>
<td>IIF RadioMsg</td>
<td>send encoded text data via radio signal</td>
</tr>
<tr>
<td>IIF RadioVoice</td>
<td>send vocal communication modulated on radio signal (could also be encrypted in digital radio communication). Traditional voice</td>
</tr>
</tbody>
</table>
12 BEHAVIOUR

12.1 Use Case Diagrams

The following paragraphs collect the use case diagrams (UCD) describing the behaviour of the CUS in the ERS and derived from the SoSUCM of Figure 3.

12.1.1 Process Emergency 1 (CUS viewpoint)

![Figure 41 Process Emergency 1 (CUS viewpoint) UCD](image-url)
12.1.2 Ask for First Aid Rescue (caller viewpoint)

Figure 42 Ask for First Aid Rescue UCD

12.1.3 Process First Aid Rescue (caller viewpoint)

Figure 43 Process First Aid Rescue UCD
12.2 Activities
The following sections collect the UML activity diagrams of the Use Case under study.

12.2.1 Process Call

Description: This activity diagram refers to the Receive emergency call use case and to physical components shown in [block] CUS Phone Answering internal block diagram.
It is quite complex as it deals with improper calls, calls to be diverted to other organizations, calls requiring the ERU intervention or not, calls referring to casualties already being handled due to previous calls.

All these activity paths include the call queuing, the call extraction from the queue by an available operator, the automatic recording and the insertion of information by the operator.

The choice of including all possible paths in the diagram is to highlight that the current workflow in the central is very flexible in order to face its scheduling needs, continuously changing according to the events being handled (location, priority, equipment, concurrency, ...).

This diagram lacks some nodes to be further detailed as sub-activities such as Obtain casualty details according to a protocol, Locate position of Target on map, Obtain further detail, Integrate response event detail with information from a subsequent Incoming Call.

The capability of handling all possible event flows is of great importance to provide the most reliable and efficient scheduling of the Emergency Response. A modelling challenge to be investigated is the possibility for a high-ResponsePriority event to give rise to the Emergency Response while the Incoming Call is still being processed with Obtain further detail activity.
12.2.2 Answer call

Description: The Answer call is a sub-activity of Process Call. The diagram shows the physical parts (blocks) involved in the interaction.

Create Call info Bag can be invoked by operator as well as triggered by signals to be added in IIF CT Phone call events.

![Answer call Activity Diagram](image-url)
12.2.3 Obtain casualty details

Figure 46 Obtain casualty details Activity Diagram

Description: the diagram details the Obtain casualty details sub-activity of the Process Call and Obtain further detail activities. This is a draft of the protocol followed by the CUS operator during an Emergency call and the caller interview. Some operations may occur in parallel (for example looking up the location on the map while continuing the interview for other details).
12.2.4 Get target location info

Integrate response event detail

: CUS

: Save call info

: Retrieve previous call

: Start editing call info

open UI for modification of retrieved call info

may consist of choice from recent calls prompted to the operator or browsing calls by appropriate keys followed by selection

close Call info bag marking it as duplicate call; phone conversation goes on

TryGet address from phone number

: Locate position

: coordinates

Figure 47 Get target location info Activity Diagram
Description: sub activity for **Obtain casualty details**.

Before stepping into **Locate position**, the CUS operator asks the caller to provide the (approximate) target.

**TryGet address from phone number** may involve a lookup on server, with architectural details to be investigated. This can be relevant at the SoS level (e.g., will the directory be present in local or accessed in the Phone System, or through a network?)

### 12.2.5 Locate position

**Figure 48 Locate position Activity Diagram**

Description: In the baseline CUS, this activity is recalled by **Get target location info** which is sub-activity of **Process Call, Obtain casualty details**.

This is a quite general task and if the coordinates of the target location were provided, it could be fully automated. With the proper onboard radio devices and peer-to-peer connection of the Radio System to the CUS Map provider, this would allow the automatic update of the ERU position.
Get coord. from toponym requires lookups on server, with architectural details to be investigated. This can be relevant at the SoS level (e.g., will the directory be present in local or accessed in the Phone System, or through a network?)

12.2.6 Obtain further detail

**Figure 49 Obtain further detail Activity Diagram**

Description: this diagram details the Obtain further detail activity in the Process Call activity.

This very simple diagram highlights the fact that handling a call or communication that was either put in stand-by or refers to an already active Emergency Response, may pass through most of the operations in the protocol defined by Obtain response details.
12.2.7 Integrate response event detail

Description: This diagram details the Integrate response event detail activity, part of the Process Call activity, triggered when a new call provides additional information to an Emergency Event already handled by an Emergency Response.
12.2.8 Replay of recorded conversation

**Figure 51 Replay of recorded conversation Activity Diagram**

Description: this diagram highlights how replaying a conversation requires lookup of conversation (automated if in recent calls) and proper authentication with a role enabled to this operation.
12.2.9 Initiate Response

Description: After a call has been evaluated as requiring a response, the Response event is initiated.

A log is created to report the subsequent events and attached to the Call info bag. All relevant information is collected while processing the call. If a Hospital is needed, the Operator verifies its availability for reception and books the admission, (taking into account the distance from the target location and the Hospital facilities).

If technical support from other organizations (this may include Police, Forest Rangers, Fire Brigades and so on) is needed, they are alerted. The activity of external organizations and their missions prosecutes in autonomous way: an interaction of ERU with ERUs from other organizations is possible (in this case like an acknowledged SoS) and may affect only the 'Attend Target' activity. According to the event details, one or more missions are started (see Initiate Mission activity). This activity ends saving the log.

Figure 52 Initiate Response Activity Diagram
12.2.10 Update response

Figure 53 Update response Activity diagram

Description: The entry point Update Response is triggered in the Process Call activity after an Obtain further detail activity: it consists in various task modifications that can be applied with no specified priority (represented in parallel): this can be performed by various operators interfacing with the same Emergency Response.
12.2.11 Initiate Mission

Description: The entry point is the Start mission signal sent in the Initiate Response activity.

The major complexity of the activity resides in allocating an ERU to the mission. In the most simple flow, an idle ERU with proper equipment is available. Otherwise, depending on priorities, the mission starts when an ERU completes or immediately terminates its mission (with respectively higher or lower priority). The scheduling choices require the knowledge of the status and location of the ERUs, thus interfacing with centrally managed data in the CUS collected by mission monitoring (see Monitor Response activity diagram).

This Initiate Mission activity is subject to failure of the radio system connecting the CUS with ERUs diversion.

Figure 54 Initiate Mission Activity Diagram
12.2.12 Radio Send message to ERU

![Radio Send Message to ERU Activity Diagram](image)

Description: this diagram details the communication from the CUS to the ERU as a 'call behaviour action' Radio Send Message to ERU in the Initiate Mission activity diagram.
12.2.13 Service Mission

**Figure 56 Service Mission Activity Diagram**

Description: The entry point Service mission signal is sent in the Initiate Mission activity. In future analyses, the Attend target1 operation may be worth detailing because meaningful at the SoS level, requiring interaction of the ERU with other possible External Organizations.
The **Acknowledge status change** activity is detailed in the **Change status** activity diagram as it involves changing status (location, ...) and reporting the change to the CUS central.

### 12.2.14 Change status

![Change status Activity Diagram](image)

**Figure 57 Change status Activity Diagram**

Description: This activity involves changing status (location, ...) and reporting the change to the CUS central.

It ends by raising the **Update ERU status** signal for **Monitor Response** activity to be started.
### 12.2.15 Monitor Response

**Figure 58 Monitor Response Activity Diagram**

Description: The entry point **Update ERU status** signal is raised whenever an ERU changes status, as detailed in the **Change status** activity (Acknowledge status change). In this case the probability of failure is greater than starting mission, but the importance of proper handling is lower.

*Process message* operation still needs detailing, being analogous to **Radio Send message to ERU** but in the reverse direction. It could trigger an automatic update of ERU coordinates in the **Map provider** of CUS.
12.2.16 Time Mission

Figure 59 Time Mission Activity Diagram

Description: the diagram shows the processes occurring in parallel to the Service Mission, with the purpose of logging time for system statistics and check against legal time limit.
12.3 Sequence Diagrams

12.3.1 SoS level

Figure 60 SoS level Sequence Diagram

12.3.2 Process call

Description: the Diagram here shows the most common flow among those in the Process Call activity diagram.

12.3.3 update ERU status - base Monitor Response

Figure 62 Monitor Response Sequence Diagram
Description: in the baseline system, the message is queued and processed by the **Human Operator** who logs in UI (IIF Mission UI).

In the improved system with peer-to-peer radio system to CUS server, using IIF acquire **Datcom data** will not require human intervention and queuing may be omitted.

### 12.3.4 Update ERU status - baseline (Monitor Mission)

*Figure 63 Update ERU status - baseline (Monitor Mission) Sequence Diagram*

Description: This sequence is an alternative representation of the **update ERU status - base Monitor Response** sequence diagram.

It shows the baseline interaction to update the ERU status (see **update ERU status - baseline (Monitor Mission)** sequence and **Monitor Response** activity diagrams), triggered at events such as **Attend target** in **Service Mission**.

**Enqueue status message** is shown as called directly from **Radio System**, whilst in the **update ERU status - base Monitor Response** sequence diagram it was regarded as internal to CUS and preceded by a send data operation. It is still necessary to investigate which is the best representation.

In normal operation, **update ERU status** consists of updating the information about the ERU status by intervention of a CUS **Human Operator**. The **ERU** status messages are enqueued by the CUS. The CUS may request to check the ERU status in particular circumstances—e.g., to verify if a diversion to a new mission with higher priority is possible; this is shown in the 'else alt' block.

The activity ends by manually updating the **ERU** status information.

Most frequently, possible failures regard radio broadcasting.
12.3.5 Update ERU status - FT (Monitor Mission)

![Sequence Diagram](image)

Description: This sequence shows a Fault Tolerant interaction to update the ERU status (see Monitor Response activity diagram), triggered at events such as Attend target in Service Mission.

Fault tolerance is obtained with the insertion of a Mobile Phone System in the SoS.

Most frequent failures regard the radio broadcasting: for this, here is omitted the voice communication of the status via Radio System, represented in the update ERU status - baseline (Monitor Mission) sequence diagram, and an acknowledgement message is instead introduced (that could be automated in the CUS when status message is placed into message queue).

The acknowledgement message allows to detect failure (in case of missing acknowledgement within time) and provide status information to the CUS by means of the Mobile Phone System. Subsequent improvements may rely in automatic processing of status messages in the queue, bypassing the CUS Human Operator.

12.3.6 Update ERU position - baseline (Monitor Mission)

![Sequence Diagram](image)

Description: This sequence shows the baseline interaction to update ERU geographical location (same term as 'position'). It consists of updating the position by intervention of CUS Human Operator, either on ERU’s status messages (see Monitor Response activity
diagram, when Attend target in Service Mission or asking for voice communication in order to provide location detail (this may happen if it is important to communicate the position (road diversions, stuck in traffic, mechanical failure). Could happen also when solicited by Human Operator of CUS: this latter situation may occur when planning a diversion to a new higher priority mission, but should require further diagram where communication request should start from the CUS. This is different than update ERU status - baseline (Monitor Mission) sequence diagram in that to communicate a particular position, it is not sufficient to send a status message with standard encoding. Activity ends by manually updating the ERU's geographical location in the map.

12.3.7 update ERU position - FT (Monitor Mission)

Figure 66 update ERU position - FT (Monitor Mission) Sequence Diagram

Description: This sequence shows a Fault Tolerant interaction to update the ERU position (see update ERU position - baseline (Monitor Mission), obtained with the insertion of a Mobile Phone System in the SoS.

The baseline SoS interaction is represented in alt and refers to updating the position by intervention of CUS Human Operator.

In case of missed acknowledgement, it is possible, to attempt a voice communication from the ERU to the CUS, in the same way as in the baseline SoS (alt b). The Broadcast communication/voice of Mobile Phone System could be replaced by properly encoded SMS. The activity ends in the same way as update ERU position - baseline (Monitor Mission), by manually updating the ERU geographical location in the map.
12.3.8 Update ERU position - improved (Monitor Mission)

Figure 67 Update ERU position - improved (Monitor Mission) Sequence Diagram

Description: This sequence shows an improved interaction to update ERU position.

The baseline (see update ERU position - baseline (Monitor Mission)) is represented in alt 1 and a first Fault Tolerant improvement corresponds to alt 3 (update ERU position - FT (Monitor Mission)): updating the position by intervention of CUS Human Operator makes this operation discontinuous and prone to errors. In the baseline SoS, an update of the ERU geographical location happened only in particular events.

With the improved SoS, peer-to-peer communication to the Map provider (intermediated by the Radio System) allows continuous and seamless updating of coordinates transmitted by Datcom (or other similar device). An acknowledgement return message is provided to notify the reception of the new position. If the ERU does not receive such a signal within a reasonable time, alt 2 is performed, transmitting the position via Mobile Phone System.

In case of missed acknowledgement for the second transmission of coordinates within a reasonable time (e.g. due to software problems in the CUS), an app on the ERU mobile phone can highlight an error situation. If it is important to communicate the position (road diversions, stuck in traffic, mechanical failure), it is possible, then, to attempt a voice communication from the ERU to the CUS, in the same way as in the baseline SoS (alt 3), either via the Radio System (alt 3.1) or the Mobile Phone System (alt 3.2). Notice that in alt 3.1, in order to queue the ERU call request a broadcast data operation is performed, although the broadcast data already failed in alt 1. However, if alt 1 and alt
2 both failed, we can assume that the failure is due to the automatic updating process in CUS (v. Map provider part) rather than to issues with the data broadcasting via Radio or Mobile Phone network. The broadcast data operation in 3.1 sends much simpler data (call request, no coordinates or status information) so that, unless the signal is shaded, there is no reason to doubt that it will work.

12.3.9 Internal behaviour

![Internal behaviour Sequence Diagram](image)

Figure 68 Internal behaviour Sequence Diagram
### 12.3.10 Internal behaviour using processes

<table>
<thead>
<tr>
<th>Process Call</th>
<th>Initiate Mission</th>
<th>Service Mission</th>
<th>Time Mission</th>
<th>Monitor Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receive and process emergency call</td>
<td>Start response</td>
<td>Service response</td>
<td>Update ERU status</td>
<td></td>
</tr>
<tr>
<td>assess needed resources and notify to external systems</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find and allocate ERU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>par Despatch and monitor ERU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERU services Casualty handler</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report progress to Call Centre &quot;CUS&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>also par</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start Casualty handler timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and par</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>par Attend the target and update status</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attend target</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>also par</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stop Casualty handler timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and par</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drive to hospital</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return to parking bay</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 69 Internal behaviour using processes Sequence Diagram**