Designing for adaptability and evolution in system of systems engineering

Prototype III
D_8.5

<table>
<thead>
<tr>
<th>Deliverable Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature</td>
</tr>
<tr>
<td>Project</td>
</tr>
<tr>
<td>Contract Number</td>
</tr>
<tr>
<td>Deliverable ID</td>
</tr>
<tr>
<td>Date</td>
</tr>
<tr>
<td>Status</td>
</tr>
<tr>
<td>Version</td>
</tr>
<tr>
<td>Contact Person</td>
</tr>
<tr>
<td>Organisation</td>
</tr>
<tr>
<td>Phone</td>
</tr>
<tr>
<td>E-Mail</td>
</tr>
</tbody>
</table>
AUTHORS TABLE

<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
<th>E-Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uri Shani</td>
<td>IBM</td>
<td><a href="mailto:shani@il.ibm.com">shani@il.ibm.com</a></td>
</tr>
<tr>
<td>Christoph Etzien</td>
<td>OFFIS</td>
<td><a href="mailto:Christoph.etzien@offis.de">Christoph.etzien@offis.de</a></td>
</tr>
<tr>
<td>Leonardo Mangeruca</td>
<td>ALES</td>
<td><a href="mailto:leonardo.mangeruca@utsce.utc.com">leonardo.mangeruca@utsce.utc.com</a></td>
</tr>
<tr>
<td>Massimo Baleani</td>
<td>ALES</td>
<td><a href="mailto:massimo.baleani@utsce.utc.com">massimo.baleani@utsce.utc.com</a></td>
</tr>
<tr>
<td>Francesca Stramandinoli</td>
<td>ALES</td>
<td><a href="mailto:francesca.stramandinoli@utsce.utc.com">francesca.stramandinoli@utsce.utc.com</a></td>
</tr>
<tr>
<td>Valerio Senni</td>
<td>ALES</td>
<td><a href="mailto:valerio.senni@utsce.utc.com">valerio.senni@utsce.utc.com</a></td>
</tr>
<tr>
<td>Marco Marazza</td>
<td>ALES</td>
<td><a href="mailto:marco.marazza@utsce.utc.com">marco.marazza@utsce.utc.com</a></td>
</tr>
<tr>
<td>Benoit Boyer</td>
<td>INRIA</td>
<td><a href="mailto:Benoit.Boyer@inria.fr">Benoit.Boyer@inria.fr</a></td>
</tr>
<tr>
<td>Evegeny Shindin</td>
<td>IBM</td>
<td><a href="mailto:EVGENSH@il.ibm.com">EVGENSH@il.ibm.com</a></td>
</tr>
<tr>
<td>Roy Kalawsky</td>
<td>LU</td>
<td><a href="mailto:R.S.Kalawsky@lboro.ac.uk">R.S.Kalawsky@lboro.ac.uk</a></td>
</tr>
<tr>
<td>Yann Lebeaupin</td>
<td>SODIUS</td>
<td><a href="mailto:ylebeaupin@sodius.com">ylebeaupin@sodius.com</a></td>
</tr>
<tr>
<td>Mickaël Albert</td>
<td>SODIUS</td>
<td><a href="mailto:malbert@sodius.com">malbert@sodius.com</a></td>
</tr>
</tbody>
</table>

CHANGE HISTORY

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason for Change</th>
<th>Pages Affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2014-06-26</td>
<td>Initialized from D8.4</td>
<td>All</td>
</tr>
<tr>
<td>0.2</td>
<td>2014-07-10</td>
<td>Adding solutions chapter</td>
<td>New chapter 6.</td>
</tr>
<tr>
<td>0.3</td>
<td>2014-07-11</td>
<td>Adding MDWorkbench description</td>
<td>Chapter 5.3</td>
</tr>
<tr>
<td>0.4</td>
<td>2014-07-14</td>
<td>Updating tools net and SMC clients info.</td>
<td>Chapter 5 + reformat of chapter 6.</td>
</tr>
<tr>
<td>0.5</td>
<td>2014-07-17</td>
<td>Replace chapter 6 from details to references to D4.4, and adding summary chapter 6 instead.</td>
<td>Chapter 6.</td>
</tr>
</tbody>
</table>
CONTENT

1 OVERVIEW, PURPOSE AND SCOPE ................................................................. 9
  1.1 THE TECHNOLOGIES WIKI PAGE .......................................................... 11
  1.2 SOLUTIONS WIKI PAGE ................................................................. 16
  1.3 TRACEABILITY OF REQUIREMENTS (WP3) AND TOOLS ................................ 18

2 SOS MODELING TOOLS .............................................................................. 21
  2.1 RHAPSODY ............................................................................................... 21
    2.1.1 Overview ............................................................................................ 21
    2.1.2 Sharing UPDM models in the tool-net .................................................. 21
    2.1.3 Tools and environment ....................................................................... 22
    2.1.4 Demonstration scenario ..................................................................... 23
  2.2 SYSTEM ARCHITECT .................................................................................. 26
  2.3 ARCHITECTURE PATTERNS ................................................................. 26
    2.3.1 Purpose ............................................................................................... 26
    2.3.2 Tools and environment ....................................................................... 29
    2.3.3 DANSE SoS Architecture Patterns Library ......................................... 31
    2.3.4 Demonstration scenario ..................................................................... 33
    2.3.5 Future Developments ......................................................................... 36
  2.4 ARCHITECTURE OPTIMIZATION WORKBENCH ..................................... 36
    2.4.1 Purpose ............................................................................................... 36
    2.4.2 Tools and environment ....................................................................... 36
    2.4.3 Demonstration scenario ..................................................................... 37

Version | Status | Date       | Page
--------|--------|------------|-----
1.0     | Final  | 2014-08-03 | 3 of 82
3 CONSTITUENT SYSTEM MODELING TOOLS ........................................................................ 47

3.1 RHAPSODY ........................................................................................................... 47

3.2 ABSTRACTION TOOLS – STATISTICAL LEARNING ........................................... 47

3.2.1 Purpose ........................................................................................................... 47

3.2.2 Tools and environment .................................................................................... 47

3.3 MODELICA W/ SYSTEM MODELER ...................................................................... 48

4 JOINT SIMULATION AND ANALYSIS TOOLS .............................................................. 49

4.1 RHAPSODY SYSML FMU EXPORTER ................................................................... 49

4.1.1 FMU generation on Rhpsoidy .......................................................................... 49

4.1.2 Rhapsody model export................................................................................... 50

4.2 DYMOLA SIMULINK FMU EXPORT ..................................................................... 51

4.2.1 Purpose ........................................................................................................... 51

4.2.2 Tools and environment .................................................................................... 51

4.3 PLASMA STATISTICAL MODEL CHECKING ......................................................... 52

4.3.1 Purpose ........................................................................................................... 52

4.3.2 Tools and environment .................................................................................... 52

4.3.3 Demonstration scenario .................................................................................. 53

4.4 DESYRE JOINT SIMULATION .............................................................................. 53

4.4.1 Purpose ........................................................................................................... 53

4.4.2 Tools and environment .................................................................................... 53

4.4.3 Demonstration scenario .................................................................................. 54

4.5 TESTCAST TEST GENERATOR ............................................................................. 56

4.6 CONTRACT-BASED RUN-TIME VERIFICATION ....................................................... 56

4.6.1 Purpose ........................................................................................................... 56

4.6.2 Tools and environment .................................................................................... 56

4.6.3 Demonstration scenario .................................................................................. 57

4.7 GCSL CONTRACTS ANALYSIS ............................................................................ 57

4.7.1 Purpose ........................................................................................................... 57

4.7.2 Tools and Environment .................................................................................... 57
4.7.3 Demonstration scenario

4.8 FAULT-BASED TEST GENERATION

4.8.1 Purpose

4.8.2 Tools and Environment

4.8.3 Demonstration scenario

4.9 TIMING ANALYSIS

4.9.1 Purpose

4.9.2 Tool & Environment

5 SEMANTIC MEDIATION AND INTEGRATION

5.1 TOOL-NET SEMANTIC MEDIATION CONTAINER (SMC) PLATFORM

5.1.1 Purpose

5.1.2 Tools and environment

5.1.3 Demonstration scenario

5.1.4 Enhancement in Prototype III

5.2 PROTÉGÉ ONTOLOGY EDITOR

5.3 MDWORKBENCH MEDIATION RULES EDITOR

5.3.1 Purpose

5.3.2 Usage

5.4 SMC CLIENT SDK

5.5 FMU IMPORTER CLIENT

5.6 MODEL INSTANCES GENERATOR CLIENT

6 SUMMARY

7 ABBREVIATIONS AND DEFINITIONS

8 REFERENCES
Figures

Figure 1-1: Tools net for the DANSE prototype II ................................................................. 10
Figure 1-2: Tools connectivity for Prototype III ................................................................. 10
Figure 1-3: One of the work flows cases discussed in the solutions wiki page .................. 16
Figure 1-4: Evaluation of Prototype II tools (from D3.7) ..................................................... 18
Figure 2-1: UPDM Mediation MDWorkbench Support ....................................................... 22
Figure 2-2: MOF Profile/Metamodel to OWL Ontology “Productivity” Tool Chain .......... 22
Figure 2-3: UPDM Mediation instantiation into the Jazz/DM Tool-Net ........................... 23
Figure 2-4: UPDM Sharing – Preparation and Initial State ............................................... 24
Figure 2-5: MDWorkbench UPDM Data and Diagrams Storage ......................................... 24
Figure 2-6: MDWorkbench System Architect Tool-Net Connection ............................... 24
Figure 2-7: MDWorkbench Rhapsody Tool-Net Connection ............................................. 25
Figure 2-8: MDWorkbench Rhapsody Data and Diagram Import ...................................... 25
Figure 2-9: Tool-Net Constituents Sharing at different depth levels ..................................... 25
Figure 2-10: SoS optimization workflow ............................................................................ 37
Figure 2-11: Communication system technical internal block diagram ............................. 38
Figure 2-12: Optimal communication system architecture ................................................ 39
Figure 2-13 CAE SV1 view ............................................................................................... 41
Figure 2-14: Simple Rewriting Rule adding a District to the SoS model ......................... 42
Figure 2-15: Select an existing Contract ............................................................................ 44
Figure 2-16: Edit a Contract .............................................................................................. 45
Figure 2-17: Link Contract to model element ...................................................................... 46
Figure 4-1: Installing the Rhapsody plugin for semantic mediation export/import SysML models .......................................................... 50
Figure 4-2: Applying this plugin as a profile to a project .................................................... 50
Figure 4-3: Activate the export command ........................................................................ 51
Figure 4-4: Setting up user credentials and target server ................................................. 51
Figure 4-5: Statistical Model Checking tool chain in DANSE ........................................... 52
Figure 4-6: DANSE System of System Design & Simulation Workflow .......................... 54
Figure 4-7: DESYRE Import wizard ................................................................................ 55
Figure 4-8: FMU importing ............................................................................................... 55
Figure 4-9: GCSL in DANSE ........................................................................................... 57
Figure 4-10: GCSL in DANSE with CAE WayForward ..................................................... 58
Figure 4-11: Fault-based test generation within the DANSE tool-net ............................... 59
Figure 4-12: Integration of realtimeAnalyzer ..................................................................... 60
Figure 5-1: Installing the Rhapsody plugin for semantic mediation export/import SysML models as part of the DANSE profiles (Chapter 2.6). The console shows the version and initialization of this profile plugin ........................................... 64
Figure 5-2: Version identification of components of the profiles including the SMC plugin profile ................................................................. 64

<table>
<thead>
<tr>
<th>Version</th>
<th>Status</th>
<th>Date</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Final</td>
<td>2014-08-032014-08-22</td>
<td>6 of 82</td>
</tr>
</tbody>
</table>
**Figure 5-3:** Activate the export command.................................................................65  
**Figure 5-4:** Setting up user credentials and target server. ........................................65  
**Figure 5-5:** Successful export done ........................................................................65  
**Figure 5-6:** Create a new UPDM project and apply to it the DANSE plugin...............66  
**Figure 5-7:** Apply the SMC import command ..............................................................66  
**Figure 5-8:** Enter user credentials, source server IP and URI of an element to import. 66  
**Figure 5-9:** Populate a diagram from the created model for this import.....................66  
**Figure 5-10:** End result of this import .........................................................................67  
**Figure 5-11:** Installing Protege 4.3 .............................................................................67  
**Figure 5-12:** Control panel of the SMC plugin to Protege .............................................68  
**Figure 5-13:** Protege editing session with an ontology from SMC. ...............................69  
**Figure 5-14:** Example of MDWorkbench documentation ...........................................70  
**Figure 5-15:** A page from MDWorkbench semantic mediator tutorial .......................71  
**Figure 5-16:** Example SMC client to demonstrate the SDK. ........................................72  
**Figure 5-17:** Javadoc documenting the SMC client API ............................................72
Tables

Table 1-1: Outline of prototype cases described in this document.................................................................16
Table 1-2: DANSE solutions compiled into the solutions wiki page............................................................17
Table 1-3: Tracing users requirements to the prototype III technologies.....................................................19
Table 2-1: DANSE profiles components .......................................................................................................42
Table 4-1: Mapping from SysML to FMI........................................................................................................49
1 Overview, Purpose and Scope

The third DANSE prototype has extended functionality over the second prototype [48], and also a small number of additional tools and technologies. The tools are used by users for the DANSE use cases and are discussed over the DANSE web-based exploitation forum, as well as being the major item on the DANSE periodic face to face meetings. The maturity of the tools from the users’ point of view is discussed in the D3.6 [43] (Validation Report - Prototype Iteration I – delivered already at M18) and D3.7 [47] (Validation Report - Prototype Iteration II – available at final draft at the writing of this document) deliverables. Together, these documents complement the tools state and make a fuller picture of the DANSE technology and tools environment from the vendors as well as users’ viewpoints.

One important addition to the tool-net support documents is the creation of a “Solutions” wiki page in which usage scenarios of the tools within functional contexts that users can relate to. These pages together with the technologies wiki page complement useful technology information needed by the users. That task is never ending, but users are more content with the DANSE tools today than when we started with the first prototype.

The work on the tools and the documentation for using them (installation, guides, tutorials, examples, etc.) still continues and will not be done by the delivery time for this document. Yet, the document must be delivered on time, after being delayed to M33 (from M30). Realizing this never-ending task, the final status of the DANSE tools at the end of the project will constitute the fourth prototype and will be extended over the present prototype. Also, we realize that as the tools are used and experience is gained by both vendors and users, they gather new insights into the DANSE process. That is all reflected on the DANSE exploitation forum where a wish-list is also formed to hold what should lay ahead of DANSE when it comes to its end.

This document is a follow up and revision of prototype II and will repeat all information from the previous document that is still true today. Not all tools and technologies have changed and for many of them the changes are not dramatic, but rather the natural evolution and fixing of bugs or addition of some features. The document is not intended to serve a “what’s new” per each of the tools, except where the new capabilities may deserve special identification.

Listed here and briefly reviewed are tools, services and technologies that partners have worked out during the first 33 months of the project, some of which have been demonstrated in the year 1 review and again (with the new additions during the 2\textsuperscript{nd} year) also in the 2\textsuperscript{nd} period review. The work flows among the tools in a networks of integrated tools, is a target of the consortium to achieve with this (on going) 3\textsuperscript{rd} iteration, and has been reflected in the new wiki “Solutions” page(s).

The following figures render the tools and technologies connectivity, the level of integration into the tool-net platform, and showing how the tools can connect with each other to share artefacts and perform analysis and design tasks. That is complemented with the useful flows as described in the “Solutions” wiki page(s).
Figure 1-1: Tools net for the DANSE prototype II

The first Figure 1-1 shows the status of Prototype II, in which most of the DANSE technologies to-date are present, and where some of the connectivity features are not integrated into the tool-net.

The next Figure 1-2 depicts a progress into Prototype III, with more technologies, and better integration into the tool-net. The chart is busier, and demonstrates the dynamic nature of the relations between vendors and users in the DANSE project, trying to respond in a short time to changing requirements on how to use the tools, their usability, power and usefulness, and the quest to produce new solutions and new technologies.
that may be useful or that can be experimented by DANSE users and use cases as long as the project linger, and serve a basis for future endeavours.

To facilitate rapid dissemination of the technologies among DANSE partners and users, DANSE partners use wiki pages, the DANSE SVN servers, and the exploitation forum. The SVN is used to distribute documentations, tutorials, information, and executable artefacts, which are linked in the wiki pages, while the exploitation forum is used to discuss issues of users against the tools, disseminating revisions of the technologies with new features and resolved issues, and producing new ideas and follow up wishes based on the DANSE experience.

1.1 The Technologies wiki page

A wiki page (https://www.danse-ip.eu/redmine/projects/danse/wiki/Technology) combines information about all the DANSE technologies, which consist of methods, tools extensions, new tools and platforms – all of which also include off-the-shelf (OTS) products.

- The wiki page starts with a couple of tables in which some 25 identified technologies are listed with basic characteristics.
- For each technology, a short overview section which leads the user to documentation for installation, learning, demonstrations and also code availability links.
- All the documents are either on the SVN server (https://www.danse-ip.eu/svn/danse/), or other available means such as an RTC (Rational Team Concert) server maintained by IBM for the consortium members.
- The RTC server can – although not used that much thus far – be used for management, use-case follow up, and collaboration amongst the DANSE members in carrying out common collaborative tasks.

Each entry in the wiki page follows this example:

- **Purpose**: Describe in general terms the purpose of this tool/technology.
- **Hardware requirements**: With the terminology used in the table, indicate this. E.g. H/L+W/64. Possibly, provide specifics to clarify. E.g., 4-8 CPU, 8-16GB RAM, 0.5-1 TB disk, High speed network, Linux or Windows, 64 bit.
- **Intended users**: Specify levels as defined above. E.g., 1,3. Than possibly, expand on that in sub-bullets. E.g.:
  - Systems Engineer using an COTS tool.
  - IT Specialist to setup networks for use of the tool.
- **Installation and setup**: Explain where to find documentation and resources for installing and setup of the tool.
- **Tutorials**: Explain where to find tutorial presentations on the tool installation and usage.
- **Examples**: Point to examples for this tool so that users can verify proper installation.
The wiki table is presented in this document to reflect the status as of the writing of this document, with the following Table 1-1 being developed by the DANSE community to facilitate more usability with that table.

The tools are both off the shelf (OTS) tools, products which can be obtained in the open market, new tools which are developed by partners of DANSE, new tools capabilities for the OTS tools – capabilities which DANSE partners are developing, extensions of existing capabilities and features, developed in DANSE, and technologies or methods which involve any of these tools and capabilities, which are also developed in the project.

This table organizes the tools into these 4 categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SoS Modeling Tools</strong></td>
<td>Tools used to work at the SoS level</td>
</tr>
<tr>
<td><strong>Constituent System Modeling Tools</strong></td>
<td>Tools used at the constituents level, models of the constituents making up the SoS.</td>
</tr>
<tr>
<td><strong>Joint Simulation and Analysis Tools</strong></td>
<td>Analysis, simulation and optimization tools applied to the models on both constituent and SoS levels, according with the DANSE methodology.</td>
</tr>
<tr>
<td><strong>Semantic Mediation and Integration</strong></td>
<td>Tool-net interoperability, integration and collaboration platform is using semantic mediation to facilitate sharing of models mong different tools in the DANSE eco system, and automation of processing stages in the DANSE methodology.</td>
</tr>
</tbody>
</table>

The table uses the following columns to facilitate important information about the tool/technology in each row:

<table>
<thead>
<tr>
<th>Column</th>
<th>Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tool Form</strong></td>
<td>Indicates if the tool is OTS, independent tool, or part of another tool, being an extension or new capability to that tool.</td>
</tr>
<tr>
<td><strong>Also Requires</strong></td>
<td>Indicates which other tools are required for using it.</td>
</tr>
<tr>
<td><strong>Integrated by</strong></td>
<td>The tool/technology being integrated into the tool-net, or other facilities to facilitate participation in the DANSE development methods.</td>
</tr>
<tr>
<td><strong>User level</strong></td>
<td>The tool-net identifies 4 levels of users, specifically when integrating into the tool-net where tool interoperability and collaboration capabilities are supported:</td>
</tr>
</tbody>
</table>

**Engineer**

Systems engineer level, somebody who perform system design, working with some modeling or analysis tool. A user is expert in the tool, yet is also trained with the tool extensions per tool-net integration where collaboration and tool interoperability capabilities are important to execute DANSE methods.

**Power User**

An expert. We identify the tool-net expert, and experts in other
technologies who needs to consult the Engineer. In the tool-net, that is a user who can manage the tools interoperability on the tool-net collaboration platform. As is described here, the technologies of configuration and ontology authoring are necessary training for that user, who helps the Engineer to perform his job with respect to the network of tools. In other cases, that indicates that the technology and/or the tool are not obvious and require assistance.

**IT Manager**  
In the distributed collaborative environment of the tool-net, IT is an important aspect with internet accessibility and IP protection, firewalls and the like concerns must be managed and configured.

**Vendor**  
Tool vendor is a developer of new tools integration into the tool-net ecosystem of collaboration and interoperability. There are some technologies which are intended for such users of the tool-net to help in working with the tool-net platform protocols.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Capabilities</th>
<th>Tool Form</th>
<th>Also Requires</th>
<th>Integrated by</th>
<th>Intended User</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SoS Modeling Tools</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rhapsody</td>
<td>Represent SoS architectures using rich set of UPDM diagrams</td>
<td>Standalone and tool-net</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>System Architect</td>
<td>Represents SoS architectures using NAF, and integrated with the Rhapsody UPDM through semantic mediation on the tool-net platform.</td>
<td>Standalone and tool-net</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>Architecture Patterns</td>
<td>Create or modify UPDM models with documented patterns from other SoS and system implementations</td>
<td>Plug-In to Rhapsody + tool-net</td>
<td>Rhapsody</td>
<td>Rhapsody and Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>Architecture Optimization Workbench</td>
<td>Generate and optimize architecture alternatives based on variations from a prescribed structure</td>
<td>Plug-In to Rhapsody</td>
<td>Rhapsody, MS Excel, CPLEX</td>
<td>Rhapsody</td>
<td>Engineer, Power user</td>
</tr>
<tr>
<td>Architecture Generation with Graph Grammars</td>
<td>Automatic generation of architecture variants using graph grammar</td>
<td>Plug-In to Rhapsody</td>
<td>Rhapsody, GROOVE</td>
<td>Rhapsody</td>
<td>Engineer, Power user</td>
</tr>
<tr>
<td>DANSE modeling extension</td>
<td>Repository of various UPDM profiles for architecture patterns, architecture optimization, GCSL, Rhapsody</td>
<td>Rhapsody Profiles</td>
<td>Rhapsody</td>
<td>Rhapsody</td>
<td>Engineer</td>
</tr>
<tr>
<td>Tool</td>
<td>Capabilities</td>
<td>Tool Form</td>
<td>Also Requires</td>
<td>Integrated by</td>
<td>Intended User</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------------------------</td>
<td>---------------</td>
<td>-----------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>profiles</td>
<td>extensions, simulation, etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GCSL Editor</td>
<td>Create goals and contracts statements in GCSL with syntax checking</td>
<td>Plug-In to Rhapsody</td>
<td>Rhapsody</td>
<td>Rhapsody</td>
<td>Engineer, Power user</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Constituent Systems Modeling Tools</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rhapsody</td>
<td>Represent SoS architectures using SysML diagrams</td>
<td>Standalone and tool-net</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>Abstraction tools</td>
<td>Abstract system models to create suitably simplified models for use with SoS analysis</td>
<td>Concepts</td>
<td>None</td>
<td>None</td>
<td>Engineer, Power user</td>
</tr>
<tr>
<td>Modelica w/ System Modeler</td>
<td>Connect Modelica models into the Tool-Net semantic mediation for use as system models in an SoS analysis</td>
<td>Standalone</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td></td>
<td><strong>Joint Simulation and Analysis Tools</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rhapsody</td>
<td>Export SysML statecharts from Rhapsody as FMU archive</td>
<td>Plug-In</td>
<td>Rhapsody</td>
<td>FMU URI</td>
<td>Engineer</td>
</tr>
<tr>
<td>SysML FMU exporter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dymola Simulink FMU exporter</td>
<td>Export executable elements of a Simulink model as FMU archive</td>
<td>Plug-in</td>
<td>Simulink</td>
<td>FMU URI</td>
<td>Engineer</td>
</tr>
<tr>
<td>PLASMA statistical model checking</td>
<td>Perform statistical model checking on the results of a simulation</td>
<td>Standalone</td>
<td>DESYRE</td>
<td>DESYRE</td>
<td>Engineer, Power user</td>
</tr>
<tr>
<td>DESYRE joint simulation</td>
<td>Perform joint simulation of multiple SoS and system models available in FMU format</td>
<td>Standalone</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>TestCast test generator</td>
<td>Generate tests for the TTCN-03 and MBT standards with semantic mediation in the Tool-Net</td>
<td>Standalone</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>Synthesis for Diagnosis and Prognosis: Contract-based run-</td>
<td>Generate executable monitors for joint simulation from GCSL contracts Synthesize executable monitors from GCSL contracts; run joint simulation with run-time property verification</td>
<td>DESYRE Analysis</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>Tool</td>
<td>Capabilities</td>
<td>Tool Form</td>
<td>Also Requires</td>
<td>Integrated by</td>
<td>Intended User</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td>-----------</td>
<td>---------------</td>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>time verification</td>
<td>Formal verification of GCSL Contracts relations (Dominance)</td>
<td>DESYRE Analysis</td>
<td>DESYRE</td>
<td>Tool-Net</td>
<td>Engineer, Power User</td>
</tr>
<tr>
<td>GCSL Contracts Analysis Tool</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fault-based Test Generation Tool</td>
<td>Automatic Test Suite Generation to verify SoS robustness to the occurrence of faults.</td>
<td>DESYRE Analysis</td>
<td>DESYRE</td>
<td>Tool-Net</td>
<td>Engineer, Power User</td>
</tr>
<tr>
<td>Timing analysis</td>
<td>Formal verification of timing properties in the joint simulation</td>
<td>Stand alone</td>
<td>Rhapsody</td>
<td>None</td>
<td>Power user</td>
</tr>
</tbody>
</table>

### Semantic Mediation and Integration

<table>
<thead>
<tr>
<th>Tool</th>
<th>Capabilities</th>
<th>Tool Form</th>
<th>Also Requires</th>
<th>Integrated by</th>
<th>Intended User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool-Net Semantic Mediation Container (SMC) Platform</td>
<td>Dynamic connection of information among tools using semantic mediation</td>
<td>Standalone Jazz server w/ extensions</td>
<td>None</td>
<td>None</td>
<td>Power user, IT admin</td>
</tr>
<tr>
<td>Protégé ontology editor</td>
<td>Edit and/or create ontologies that define the semantic mediation for use in Tool-Net Semantic Mediation Container</td>
<td>Plug-in to Tool-Net</td>
<td>None</td>
<td>Tool-Net</td>
<td>Power user</td>
</tr>
<tr>
<td>MDWorkbench mediation rules editor</td>
<td>Creation of rules for UPDM models mediation between Rhapsody UPDM, a common UPDM, and NAF (System Architect)</td>
<td>Standalone</td>
<td>None</td>
<td>Tool-Net</td>
<td>Power user</td>
</tr>
<tr>
<td>SMC client SDK</td>
<td>Software Development Kit (SDK) for developing SMC clients by tool vendors.</td>
<td>Standalone</td>
<td>None</td>
<td>Tool-Net</td>
<td>Vendor</td>
</tr>
<tr>
<td>FMU Importer Client</td>
<td>Client tool to import FMU interface specifications as an RDF model</td>
<td>Standalone</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>Instance Generator</td>
<td>Client tool to generate multiple instances of model elements using</td>
<td>Standalone</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
</tbody>
</table>
### 1.2 Solutions wiki page

Following the problems raised by users who could not find sufficient information in the technology page, a solutions page which reflects on the DANSE methodology has been created ([https://www.danse-ip.eu/redmine/projects/danse/wiki/Solnmethods](https://www.danse-ip.eu/redmine/projects/danse/wiki/Solnmethods)), and in which experience from the Conceptual Alignment Example - and some initial experience gained from the use cases – are presented as stories. These pages reflect contents that have been compiled as the DANSE methodology in the D4.3 deliverable (M24) [49], and which will have a follow up in D4.4 at the end of the project.

The work-flows in this wiki page are depicted in the next [Figure 1-3](https://www.danse-ip.eu/redmine/projects/danse/wiki/Solnmethods).

---

**Figure 1-3:** One of the work flows cases discussed in the solutions wiki page.

The following [Table 1-2](https://www.danse-ip.eu/redmine/projects/danse/wiki/Solnmethods) lists the solutions presently described in this wiki page.
### Table 1-2: DANSE solutions compiled into the solutions wiki page.

<table>
<thead>
<tr>
<th>Nbr</th>
<th>Solution Method</th>
<th>When to Use It</th>
<th>What to Expect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Model SoS</td>
<td>When the real SoS is too complex to understand directly, or when it is necessary to try issues that would be inappropriate in the real SoS</td>
<td>A resulting architecture in UPDM (primary), NAF, or SysML frameworks that closely matches the structure and behaviour of the SoS.</td>
</tr>
<tr>
<td>2</td>
<td>Abstract CS model</td>
<td>When a constituent system has its own model in modelling tools that are either not compatible with or too complicated to work with the SoS model</td>
<td>A simpler representation of the CS in a modelling tool that is compatible with the SoS model.</td>
</tr>
<tr>
<td>3</td>
<td>Apply architecture patterns</td>
<td>When significant changes are needed to the SoS in order to accomplish perceptual improvements</td>
<td>Selection and implementation of UPDM architecture patterns that have been proven to provide specific improvements in other SoS</td>
</tr>
<tr>
<td>4</td>
<td>Generate architecture alternatives</td>
<td>When variations in the basic architecture are needed that meet specific rules, but it is uncertain what variations might be possible</td>
<td>Creation of many architecture alternatives based on defined rules, along with evaluation of those alternatives against requirements</td>
</tr>
<tr>
<td>5</td>
<td>Generate optimized architectures</td>
<td>When variations in architecture quantities or connections might create configurations that better meet quantifiable requirements</td>
<td>Generation of multiple architecture alternatives, along with selection of the alternative(s) that best meet the optimization criteria</td>
</tr>
<tr>
<td>6</td>
<td>Perform joint simulation</td>
<td>When it is desired to execute the SoS and CS models in a simulation environment to observe behaviour and to evaluate performance</td>
<td>Time-based execution of the joint models, with tracking of defined model parameters and behaviours.</td>
</tr>
<tr>
<td>7</td>
<td>Perform statistical model checking</td>
<td>When it is desired to statistically check model performance against defined parameters, goals, or contracts.</td>
<td>Identification of performance levels against parameters/goals and time-based compliance against defined contracts.</td>
</tr>
<tr>
<td>8</td>
<td>Evaluate emergent behaviour</td>
<td>When it is desired to observe and evaluate the SoS emergent behaviour.</td>
<td>Confirmation or denial of known emergent behaviours against time; discovery of new emergent behaviours</td>
</tr>
<tr>
<td>9</td>
<td>Evaluate goals and contracts</td>
<td>When it is desired to define and evaluate SoS/CS performance against formally defined goals and contracts</td>
<td>Clarification of goals/contracts; knowledge of time-based compliance of an architecture against those goals/contracts</td>
</tr>
<tr>
<td>10</td>
<td>Perform formal verification</td>
<td>When it is desired to confirm that the SoS meets specific requirements.</td>
<td>Clarification of requirements in the form of contracts; knowledge of time-based compliance of an architecture against those contracts under formally defined conditions</td>
</tr>
<tr>
<td>11</td>
<td>Configure DANSE Tool-Net</td>
<td>Prior to performing any of the DANSE methodology, or to improve the environment for better use</td>
<td>Installation of necessary tools, ontologies, rules, and clients to perform DANSE methodology</td>
</tr>
</tbody>
</table>

Each solution comes with the following sub sections:

- **Description** – General description of the solution.
- **Initial situation** – A possible initial state of the problem to be solved.
- **Expected Results** – A possible final state of the problem solution.
- **Required Tools** – which of the tools and DANSE technologies are used in the solution, including possibly a diagram of the tools and their relations.
- **Activities** – Possibly a flow chart of the activities in the solution.
- **Limitation** – known limitation of the solution.
- **Follow-up Solution Methods** – which other DANSE solutions in this collection would commonly follow that solution.

This report includes also the description of the solutions following the presentation of the technologies sections.
1.3 Traceability of requirements (WP3) and tools

Reflecting on the first prototype in comparison with the technical requirements has been reported in D3.6 [43] following prototype I, and in D3.7 [44], following prototype II. In the following Figure 1-4, the tools evaluation has improved as per the users evaluation following prototype iteration II.

A traceability of these with technologies introduced into the DANSE prototype including this deliverable is presented in the next table.

<table>
<thead>
<tr>
<th>#</th>
<th>Prototype I evaluation topic</th>
<th>Status Prototype II</th>
<th>Actual use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UPDM Improvements</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>SoS domain metamodel</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Methodologies for handling dynamicity</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Methodologies for handling emergent properties</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Methodologies for handling conflicting goals</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>Abstraction techniques</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>Reusable architectural and interaction patterns</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>GCSL: Goal and Contract Specification Language</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>Generation of architecture variants</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>Architecture optimisation techniques</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>Joint simulation</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>Statistical model checking</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>Formal verification</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>Synthesis for diagnosis and prognosis</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>Automatic test case generation</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>Model mapping for model sharing</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>Sos tool interoperability infrastructure (toolnet)</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Figure 1-4:** Evaluation of Prototype II tools (from D3.7)

The assessment is according to the following convention:

- 0: poor
- 1: average
- 2: good
- 3: excellent
Table 1-3: Tracing users requirements to the prototype III technologies.

Among the questions raised by D3.6 which remain open:

<table>
<thead>
<tr>
<th>Issue</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the relation between “profiles” (tool independent part of UPDM extensions) and “plugins” (related to a profile, but possibly tool dependent)</td>
<td>In Rhapsody, a “profile” is an extension mechanism for users of the tools. A profile may consist of stereotypes, and/or it may include also code extensions. In the later case, we term this “plugin” as it uses the Java plugin API of Rhapsody. In our prototype, architecture patterns, as well as the concise modeling method for the optimization workbench are purely based on stereotypes extensions. In the FMU export and model sharing through the semantic mediation container – these are actual plugins which provide new functional capabilities to the tool.</td>
</tr>
<tr>
<td>The role of the OSLC server</td>
<td>OSLC [6] is a standard for tools interoperability which is adopted in DANSE. The semantic mediation container provides OSLC services to models that are stored in its repositories. The GCSL editor tool is also using OSLC through a special server written as an extension to Rhapsody (yet, working through a similar mechanism to plugins – through not installed as such).</td>
</tr>
<tr>
<td>The scope and extent of “concise profile” (shared between architectural patterns extension and optimization extension)</td>
<td>The “concise profile” is a profile which extend Rhapsody with new stereotypes. In addition, this extension comes with an additional functionality that processes a model decorated with these stereotypes, and submit the results into a CPLEX optimization engine. This technology is applied to select optimized architecture alternatives, which in-turn, may come from applying different architecture patterns as provided by WP5, and stored as annotated models (with the architecture patterns stereotypes extension profile).</td>
</tr>
</tbody>
</table>
2 SoS Modeling Tools

2.1 Rhombus

2.1.1 Overview

This is an off-the-shelf tool which is extensively used in DANSE and selected as the backbone of SoS modeling. For the SoS level design, DANSE adopts UPDM, and applies some extensions via profiles and stereotypes. The extension profiles of this tool include also plugins which add capabilities to Rhombus that are required in wider contexts as described per other technologies in the prototype.

As an SoS design tool, Rhombus's UPDM is considered. Yet, this tool is also used at any level of systems engineering design, including the constituent level.

In the follow-up sections, the organization of Rhombus profiles is described so that a user can associate any of the DANSE extensions to the project. Yet, this organization of profiles does not distinguish those profiles supporting design and other capabilities for any of the tools categories: SoS, Constituents, and Analysis.

2.1.2 Sharing UPDM models in the tool-net

Using Semantic Mediation as in Chapter 5, UPDM has been targeted to support the common SoS point of view for architecture definitions. But missing of standard exchange formats between heterogeneous set of tools, this has to be applied through DM based mediation. On the selected DANSE engineering workbench, it consists into mediating data between Rhombus UPDM models and System Architect UML+NAF assets. Acting as 3rd party stakeholder over the tool-net, SODIUS has implemented since many years a model “hub” based on its MDWorkbench environment. By adding a specific DANSE compliant support to its generic existing platform, it has been possible to define in a quicker, productive and unified way for several RDF/OSLC clients and providers for the Tool-Net. This preliminary work has validated in the Prototype 1 the capability of 3rd parties to enhance DANSE platform and interact deeply with the Tool-Net assets.
2.1.3 Tools and environment

First, to enable the UPDM-centric mediation on server side, a tool chain has been developed to automatically generate compatible ontology (OWL definition) from the standard UDM2 profile, enabling MOF/Ontology bridge.

For M18 implementation, goal was to bring on the shared platform all components required to iterate with users in the next steps. It fits in several components in the tool-net architecture:

1. **Wizards on Rhapsody and System Architect** clients side
2. **Complete UPDM ontology and tool mediations** deployable on server side
3. Implement first set of **mediation rules applicable System/Service levels** in the CAE: NSV1/NSV2
4. And finally, experiment sharing between **different tools with different frameworks (requiring mediation)**: UPDM for Rhapsody and NAF3 + UML in System Architect and develop related client integration
2.1.4 Demonstration scenario

The SoS is started into Rhapsody (using CEA example):

- **Postulate:**
  - A missing constituent « Communication Layer » is not yet integrated in the SoS Rhapsody UPDM models on the partner A side (integrator). A partner B is delivering abstraction of this missing constituent (provider).

- **Preparation Steps :**
  - Some manual operations can be required to prepare the model sharing between partners, abstraction for example or diagram exposing only “public” interfaces.
  - Rhapsody UPDM and System Architect NAF mediations are already deployed on the tool-net (administrator/developer roles), existing UDPM Rhapsody project and System Architect NAFv3 + UML encyclopedia are configured.
Demonstration Steps:

- A partner is responsible to provide « Communication Layer » assets from System Architect
- NAF3 models by publishing the constituent over the tool-net.

System Architect data AND diagrams are converted and stored as UPDM artifacts into the shared repository. New Available Constituents AND Views are ready-to-use into the SoS Model (from UpdmStore). Partner A
integrate them into Rhapsody using DANSE MDWorkbench UI extensions and requesting specific input ports over the tool-net.

**Figure 2-7:** MDWorkbench Rhapsody Tool-Net Connection

According the elements chosen at the System Architect sharing step, information can be shared at different levels between partners.

**Figure 2-8:** MDWorkbench Rhapsody Data and Diagram Import

**Figure 2-9:** Tool-Net Constituents Sharing at different depth levels
2.2 System Architect

This is an off the shelf tool in which NAF is implemented – a framework for SoS design. The tool is integrated into the tools-net for sharing of models with Rhapsody as described in the previous chapter.

2.3 Architecture Patterns

2.3.1 Purpose

Expression of the SoS architecture is fundamentally important when adapting or evolving a SoS to meet current and future requirements. The nature of a SoS makes conventional systems engineering approaches less useful when considering how to optimise a set of candidate architectural solutions towards a set of capability optimisation goals. The overall SoS architecture is likely to be extremely complex, comprising initially of legacy systems through to the inclusion of future constituent systems. Moreover, transition from a SoS starting state to its end state may actually go through several iterations over time. Also, poorly defined legacy systems may need to be retired and replaced in a carefully orchestrated manner along with a requirement to produce a SoS architecture that can migrate from the start to the end state taking into account key factors such as cost and timescale whilst maintaining a level of acceptable operational capability. Therefore, the manner in which the SoS architecture and its constituent systems is evolved requires a reproducible, robust and verifiable process. Architecture patterns are seen as one possible way forward in tackling the complexity associated with defining the architecture of a SoS. Architecture patterns refer to recurring structures, objects and events such that they can be used as designs, blueprints, models or templates in the construction of other structures, objects and events. When used as SoS creational elements, architecture patterns can be used as the starting point to lay basic foundations for the overarching SoS and its constituent systems. It is important to note that architecture patterns are not prescriptive, but suggestive by including guidance on when their use is most appropriate and provides examples from existing systems. Consequently a pattern has structural and dynamic properties whose form is realized through a finite number of visible and identifiable components. A component in this context can be technical or non-technical entities, services or even software. It is important to note that architecture patterns are hierarchical in the sense that high-level abstract patterns can be evolved into lower level patterns that more specifically represent the implementation form of the components of a SoS.

Generating a SoS architecting can start from many points within the evolutionary lifecycle of a SoS as shown in Figure 2-10. Also, we can consider a SoS as a continuum from the macroscopic SoS level through to the constituent systems. This might seem to be an absurd level of detail but we are dealing with interactions that can, and do, reveal themselves at different levels of the system representation framework. This is where SoS architectural patterns help with the subsequent analysis of the evolving SoS [44].
Architecture patterns are capable of being applied at different stages while designing a SoS. Figure 2-11 shows where in the SoS development process the architecture patterns could be applied.

Designing a SoS is a complex process requiring many iterations to find out the best possible solution. For a less experienced systems architect it is difficult to identify and use all the possible architecture patterns suitable to a particular domain. Thus a pattern repository [45] has been established (these patterns include pre-existing patterns mined from the original ensemble of legacy systems making up the SoS, or from patterns mined in other domains but of which are relevant in the current application) to store patterns that will help solve problems encountered when modifying the SoS. The patterns repository offers a list of patterns mined from different domains and can be applied to other SoS. Architecture patterns in the repository have been divided into two categories depending upon their occurrence and nature of use. The more frequently existing patterns that are commonly observed have been called the ‘Root Patterns’. These patterns are simple in nature and occur frequently in most SoS domains. The second category of patterns is the ‘Specific patterns’, which are context specific and have been mined from a specific scenario. Each pattern is provided with information on the background of these patterns; their domains of use, allowing the users to better understand their application. Overall the repository provides a number of alternative architectures for users to choose from and apply to SoS models.

In order to perform analysis of alternative architectural solutions contextual and relevant performance data (stored in conjunction with the pattern) can be applied to numerous optimization techniques such as Concise Modelling technology. Figure 2-11 illustrates how a SoS can be expressed in the context of an evaluation.
framework, which allows the candidate architecture SoS solutions to be evaluated through a process of design space exploration. The figure also illustrates where SoS Architecture Patterns feature in the context of design space exploration.

**Figure 2-11:** SoS Architecture Design Space Explorations and SoS Architecture Patterns

Architecture patterns can be modelled using both SysML and UPDM. Due to the large number of viewpoints available within UPDM, modelling can become difficult for the systems architect. Thus UPDM patterns have also been generated guiding the user in choosing the specific views. The patterns have been predominantly structured using Systems View (SV-1) Block Definition Diagram but are not limited to it. They can also be represented as SV-1 internal block diagram (IBD) another SysML and UPDM view, and also in operational views such as OV-5b. This is highlighted in Figure 2-12 – a simplified diagram of the DANSE methodology – where the DANSE methodology is depicted flowing through all the modelling stages through to simulation using DESYRE [23].
2.3.2 Tools and environment

SoS architecture patterns are not tools per se, instead patterns can be thought of as recurring structures, objects and events, or even a recipe that describes how to create the particular entity and the context in which it can be used. The role of patterns in supporting architecting and analysis of SoS is described above and where in the DANSE methodology they feature can be seen in Figures 2-11 and 2-12. Greater detail on
the role of patterns can be found in Section 2 of deliverable 5.2 [46]

Here, we look to create patterns within Rhapsody models, more specifically, block and part diagrams in SysML, (patterns can be created in other SysML models or at the UPDM level). Expressing SoS architectures through patterns provides system architects and designers with an opportunity to create libraries of reusable components based on prior experience or standard practices. There are three key processes involved in the use of patterns for SoS. The first is clearly the creation of patterns, followed by pattern selection and then refinement of the pattern for subsequent use/deployment within the architecture of a SoS [Figure 2-13]. Since SoS are most likely to evolve from a collection or pre-existing systems it seems logical to mine patterns by examining the legacy system to see if it possible to create a representative pattern. In fact the pattern for a legacy system may be the only artefact that can be used to represent a given system on account of its inner operational structure being inaccessible. Also, in the case of a future evolving SoS the exact form the SoS takes may be unknown at the outset but through the use of appropriate architectural patterns it may be feasible to represent the SoS so that analysis can take place. In the first instance, experienced practitioners will need to extract specific patterns since they have the knowledge of what is important (and potentially re-usable).

Figure 2-13: Six-stage process showing architecture pattern process
Moreover, the need to share patterns has been recognized in the software engineering community and large efforts have been made to identify the most suitable methods to achieve this. If we accept that SoS architects will be using different tools we need to ensure the SoS architecture patterns are available in different formats. The most basic form being a simple word document that can be read by anyone.

Consequently, the DANSE SoS Architecture Pattern library will make accessible to the architect, patterns that will help solve problems encountered when modifying the SoS. More importantly, additional data such as performance information stored as part of the architectural pattern makes it easier to model and simulate future states of the SoS with greater confidence.

2.3.3 DANSE SoS Architecture Patterns Library

The SoS Architecture Patterns Library allows intelligent pattern searches against a range of attributes, for example:

- Search: Key word searching
- Propose/edit patterns: Allowing experts to propose new patterns or edit existing patterns
- Training: providing training material and applied examples for specific patterns, allowing experts to transfer implicit knowledge to less experienced engineers
- Download: following search functions, the platform allows user to download the relevant Rhapsody model individually or as a package

For the DANSE project, and given the different institution/organisation IT requirements/constraints the SoS Architecture Patterns Library has been implemented as a searchable Microsoft Word document and an online multi-platform compatible database. The latter was provided by FileMaker, which can run natively on Apple Mac or PC platforms. In addition, for those users who do not have access to a FileMaker license it is possible to access the FileMaker database via its Instant Web Publishing Engine (IWP). The IWP allows any web browser to access the database with the minimal of fuss and does not require any additional software components to be installed by the user. This is especially important for companies who have extremely tight control over which software they allow their employees to install on their computers. FileMaker also provides a very useful Apple iPAD App that can connect to the FileMaker server. This APP is downloadable for free from the Apple Store.

Loughborough University maintains the SoS Architecture Patterns library on a dedicated FileMaker server for the DANSE project. The online library provides a powerful search engine and also the means for users to create, copy, and modify SoS architecture patterns so that these can be made available to others in the DANSE project.

There are two elements to the SoS Architecture Pattern Library each are described in more detail below:
2.3.3.1 SoS Architecture Pattern Library Document

The document library is made up of all the patterns generated to date and is frequently updated to capture new architecture patterns from the CAE and the test case scenarios.

We have decided to classify patterns as "Root Patterns" - these are generic (application independent) patterns and Specific Application Patterns" - these are derived from the Root Patterns but are tailored to specific application domains. In the fullness of time we hope to have many Specific examples derived from the CAE and the three Industrial Test Cases. Section 4.4 of Deliverable 5.2 [3] provides a examples of such patterns.

The composed library of SoS architecture patterns have been shared with all those involved in the CAE and the Test Case holders. As specific patterns become re-used it will be possible to publish a robust set of domain specialist/expert architecture patterns.

2.3.3.2 SoS Architecture Pattern Library: Web-Based Repository

A web-based SoS Architecture Patterns Library has been created (Figure 2-14) to make accessible the patterns created to date but will continue to grow as more architecture patterns are mined. The online repository presents the patterns in a similar format to that created in D_5.1 [3], but is more interactive allowing for easier navigation and search functions to find specific patterns. Important features of the database include allowing the user to export full patterns in PDF format or to download the relevant Rhapsody files (.sbs) of a pattern for their projects.
2.3.4 Demonstration scenario

The process can be illustrated on CAE test case, in which we look at the evolutionary development of the Emergency Response SoS architecture. The constituent systems of three key emergency response agencies (Police, Fire and Medical) are under review to move from a totally Tetra based communications network to an LTE network, or even a hybrid solution to support intra and inter-agency communications. Details of this scenario refer to Section 4.1 in Deliverable 5.2 [3]. The preliminary motivations for the move from Tetra to LTE are the increased data handling capabilities that LTE offers, namely image and video streaming. There is also a motive to move from a decentralised command and control emergency response
structure to a more centralised (or other) configuration pattern that supports the changes in population, technology, environment etc.

**Figure 2-15:** “CentralisedCommandControlHQAndTETRA/LTECommunicationNetwork” Pattern in Online Library

In order to incorporate an architecture pattern into the CAE test case, the architect should select patterns from our SoS Architecture Patterns Library and download the corresponding patterns models for further use. A pattern named “CentralisedCommandControlHQAndTETRA/LTECommunicationNetwork” is chosen here to illustrate patterns incorporation whilst Figure 2-15 shows this pattern’s details represented in SoS Architecture Patterns Online Library with downloadable UPDM models (UPDM model and UPDM model with concise stereotypes). By importing UPDM models into IBM Rhapsody as profiles, this pattern is ready to be applied into target System View Package. Specific procedure and results of each stage of this pattern’s incorporation can be found in Section 4.5 of Deliverable 5.2 [3]. The structure of former model incorporated
into CAE is shown on Figure 2-16. Then the SoS architect is able to add, modify and remove any element in this structure. For further architecture optimisation, apart from modifying and removing the constraints and concise stereotypes in the architecture pattern-based model, the SoS architect should add more concise stereotypes and constraints according to CAE’s test case scenario.

![Figure 2-16: Structure of the Selected Pattern’s UPDM Model Incorporation](image)

In Section 4.5 of Deliverable 5.2 [3], there is a possible initial architecture as the result of the SoS architect’s manual effort based on above architecture pattern-based model. The incorporation of the other model of this selected pattern is represented in Section 4.6 of Deliverable 5.2 [3] showing pattern’s close collaboration with Concise Modelling. Also, alternative patterns in SoS Architecture Patterns Library provide the ability of architecture trade-offs.
2.3.5 Future Developments

Consideration is currently being given to linking the SoS Architecture Patterns Library with an ontology tool so that searching through the patterns can be supported by automatics pattern searching techniques.

2.4 Architecture Optimization Workbench

2.4.1 Purpose

One of the main tasks of SoS modeling is to design SoS architecture satisfying all SoS and constituent system requirements and optimizing SoS goals as well as goals of all constituent systems. However, ever-increasing complexity of today’s systems, strict design constraints, conflicting goals, and many other factors turns process of finding optimal design to an extremely difficult task. The purpose of concise modelling and optimization technology [39] is performing a multi-objective parameterized optimization of SoS architecture from an architectural pattern in Rhapsody and a list of parameters in MS Excel, using CPLEX solver.

2.4.2 Tools and environment

The concise model consists of set of views, data schema and corresponding input data. The set of views includes requirement (functional) layer, architecture (technical) layer and mapping between these layers, and can be further extended by indexing (geometrical) layer and corresponding mapping from architecture to indexing layer. The set based on architectural pattern and can be extracted directly into Rhapsody using tool-net mechanism, by choosing specified architectural pattern from pattern repository.

The views are based on SysML [17] Rhapsody [37] model with concise profile extension. Each layer can be represented as SysML internal block diagram. Concise profile extension includes set of stereotypes used for modeling and optimization purposes. Some of these stereotypes (<<catalog>>, <<inventory>>, …) represents relationship between SysML elements and corresponding data tables, while other (<<optimized>>, <<sow_constraint>>, <<sow_goal_attribute>>, …) marks SysML elements as decision variables, optimization constraints and goals. There are also set of stereotypes used for domain specific pluggable algebras. A detailed description of concise profile can be found in D.6.5.1 [40].

The data schema represented by specially formatted Excel workbook. This workbook can be created from concise model by Rhapsody concise plug-in or extracted from repository by using tool-net mechanism. The workbook must be updated each time when corresponding model changed to keep relation between data and model (Rhapsody concise plug-in can be used for this purpose). Data tables from external sources can be copied into corresponding excel worksheets manually or automatically using various existing techniques.

Concise plug-in automatically translates concise model and data into optimization model code and run CPLEX solver [41] to obtain set of Pareto-optimal solutions. These set of solutions can be further ranked and filtered according to user preferences and automatically translated into set of back-annotated SysML models.
Concise modeling and optimization process represented in Figure 2-10: SoS optimization workflow. The process can be repeated for different architectural patterns to obtain set of optimal solutions over set of architectural patterns.

2.4.3 Demonstration scenario

The process can be illustrated on communication system use case. Communication systems and services are critical parts in system of systems and their interaction. In our use case we consider the communication system evolution. The purpose of the use case is to find the optimized solution for the transition from Tetra to LTE technology taking in consideration the changes that must be implemented on the constituent systems and maximizing the overall benefits of the new technology while optimizing the best placement of new antennas or replacement for old ones. The use case utilizes following domain specific knowledge:

- Geographical domain knowledge utilized by existing communication system antennas disposition, possible places for new antennas and maximum numbers of antennas in selected positions.
- Radio-electronic domain knowledge utilized by coverage tables, communication equipment types and possible equipment connections.
The main parts of the potential communication system topology shown in Figure 2-11: Communication system technical internal block diagram. The diagram represent following technical knowledge, requirements and constraints for communication system:

- There is coverage area that must be covered by two types of mobile networks (Area).
- There is existent Tetra network infrastructure can be reused (prevAntennaTetraInstall)
- There are 3 different types of antennas: one can be used in LTE network only (Antenna LTE), one can be used in Tetra network only (Antenna Tetra) and one can be used in both networks simultaneously (Antenna Generic).
- There are 2 different types of controllers: one capable to control LTE and Tetra antennas only and one capable to control Generic antennas only.
- Each antenna must be connected to one controller placed in command center.
- Number of antennas connected to one controller dependent on controller model.
- The coverage data for both types of mobile network provided by corresponding coverage tables.

The coverage tables as well as table describing existing Tetra network infrastructure imported into Excel workbook. There is also Excel worksheets representing catalogs of possible antenna and controller models which are including various technical characteristics.

There are also other SysML views and data describing functional and geometrical information, requirements, constraints and data.

The optimization goal is provide architecture of communication system that maximizes coverage of both communication networks minimizing system cost.

One of the optimal architectures is shown on Figure 2-12: Optimal communication system architecture. This architecture provides 98.5% coverage for both networks (which best possible coverage) by minimal cost.
2.5 Architecture Generation

Architecture generation relies on two technologies, one being the Architecture Patterns (2.2), and the use of Graph Grammars. The later has been presented in the first prototype [42]. However this technology is only in development at the time of this prototype, and will be available for use for prototype III.

2.5.1 Graph Grammar

2.5.1.1 Purpose

The graph rewriting rules (aka Graph Grammar) are used to model the dynamic reconfiguration of the SoS structure and the participating CSs. This represents the evolution of the SoS. The investigation of subsequent SoS architectures is based on UPDM (IBM Rational Rhapsody) and therefore this tool allows to generate a new architecture out of an existing one.

This implements the solution method “Generate Architecture Alternatives”.

2.5.1.2 Tools and environment

From the pure technical point of view this tool requires:

- IBM Rational Rhapsody 8.0.*
- UPDM 2.0
- DANSE Extension Profile
- GROOVE 4.9.3

The first two for modelling the SoS model, the third to model the rewriting rules and the GROOVE generator to generate the alternatives.
2.5.1.3 Demonstration Scenario

The following example is copied from the “DANSE D3.3.2 Concept Alignment Example Description” deliverable and adapted/updated for this document.

The CAE behavioural part presents the behaviour of one subset of the CAE SoS that mainly consists of ten districts of a city, several fire brigades, four fire stations and one fire head quarter. The dynamicity in this setting can be located at the assignment of fire stations to the regions, the allocation of fire brigades to fire station, an additional fire head quarter or the growth of the city in term of an increasing number of residents and/or additional districts. The concept of graph rewriting is used to describe what are the possible architectures of the SoS. This concept can be used during the design exploration phase where different possible architectures are identified and compared against each other. The other purpose is to anticipate possible future architectures which result from evolutionary change of the SoS.

In the CAE behavioural part, a design exploration scenario is to assign the responsibility of the fire stations to the districts. This assignment goes along with the challenge to ensure that the fire brigades can reach each district within a certain time bound. One intuitive first allocation is therefore that the district where a fire station is located is always assigned to this fire station. For all other districts the assignment must ensure that the distance between district and fire station is not too far. The exploration scenario is defined as if the fire stations are already deployed (which is naturally the case in existing cities) and the number of fire brigades shall be minimized by ensuring the maximal response time limit. The modelling of dynamicity allows specifying the allocation of fire brigades to fire stations. The response time could be abstracted by the number of districts between a district and the responsible fire station.

In this example we start with the SoS model illustrated in Figure 2-13.
Figure 2-13 CAE SV1 view
The evolutionary changes are represented in a rewriting rules like in Figure 2-14 where a new district is added and connected to the CAE.

![Diagram](image)

**Figure 2-14**: Simple Rewriting Rule adding a District to the SoS model

Using several of those rules the evolution of the SoS is implicitly modelled. For analysis purpose those rules and the SoS model are translated into the input language of GROOVE and the generator applies several rules which results in different SoS models. The generated SoS models are feed back into the Rhapsody tool.

### 2.6 DANSE modeling extension profiles

A collection of all Rhapsody profiles for DANSE development, organized in a convenient way for users to pick up and use in their projects. The components of this collection are relevant also to technologies listed below as they enable these technologies.

Contents of the profile:

<table>
<thead>
<tr>
<th>Technology</th>
<th>Profile Package</th>
<th>Responsible</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stochastic Annotations</td>
<td>Stochastic</td>
<td>EADS-FR</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>GCSL</td>
<td>GoalsAndContracts</td>
<td>INRIA</td>
<td>2.0</td>
<td>Removed obsolete Tags.</td>
</tr>
<tr>
<td>Simulation</td>
<td>Simulation</td>
<td>EADS-FR</td>
<td>0.1</td>
<td>Provides the “FMU” stereotype required by the “IBM FMU Plugin”</td>
</tr>
<tr>
<td>Architecture Generation</td>
<td>Dynamicity</td>
<td>OFFIS</td>
<td>2.0</td>
<td>Updated the list of connection types.</td>
</tr>
<tr>
<td>Concise Modeling</td>
<td>Optimization</td>
<td>IBM</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>SODIUS Plugins</td>
<td>ToolNet</td>
<td>SODIUS</td>
<td>2.0.2</td>
<td>enable Rhapsody &lt;-&gt; IBM ToolNet/JAZZ integration</td>
</tr>
<tr>
<td>IBM SysML and UPDM Plugin</td>
<td>ToolNet</td>
<td>IBM</td>
<td>2.16</td>
<td>Export/Import SysML models to/from the tool-net semantic mediation platform.</td>
</tr>
<tr>
<td>IBM FMU Plugin</td>
<td>Not in the DANSE profile. But built into the product.</td>
<td>IBM</td>
<td>4.0.1 extended</td>
<td>Exports FMU for SysML blocks into files or the tool-net semantic mediation platform. See “Simulation” package.</td>
</tr>
</tbody>
</table>

**Table 2-1**: DANSE profiles components
The content of these profiles are also relevant to technologies described in this deliverable.

2.7 GCSL Editor

2.7.1 Purpose

The GCSL Editor allows creating and editing GCSL statements in Rhapsody UPDM. The user is guided by the editor to select a pattern from a dropdown list and the modifiable parts of the structured text are placed as individual text fields below the dropdown list. The individual text field are check while editing for syntax errors and references to elements of the model are limited to only valid ones.

2.7.2 Tools and Environment

The GCSL Editor installer consists of the editor itself and an OSLC-based server application for Rhapsody. In addition Rhapsody 8.0 and a UPDM model with attached DANSE profile is required to use the editor. All GCSL statements are stored directly in the UPDM model and no copies are stored by the editor itself.

2.7.3 Demonstration Scenario

In the following a short example is presented where a GCSL statement is edited in a CAE Rhapsody UPDM model.

2.7.3.1 Editing a Contract

To edit the contract one can do it directly in Rhapsody or use the GCSL Editor. The benefit of using the GCSL Editor is that the list of GCSL pattern and a syntax check is integrated which is not available in Rhapsody. In order to edit an existing contract in the Editor open the UPDM model which contains the contracts you want to edit and start the click on “Open Requirement” in the Editor. A tree view of the model opens where a contract can be selected (see Figure 2-15).
In Figure 2-16 the Assumption of the Contract “No_1” is selected. In the Properties view the used “always condition” pattern is displayed. Note that with opening the dropdown menu the list of GCSL pattern is displayed and the user can select one of these. Below the “Pattern” section a “Description” and the entry for the “condition” is shown. The “Pattern Properties” section depends on the selected “Pattern” and the user must only “fill the holes” of the pattern with the specific content. This content is automatically checked for correctness of the syntax.
Any changes of the contract are stored in the Rhapsody model if the user “saves” them.

To distinguish between global and local contracts the user have to select a “Anchored Element” in the Feature Dialog within Rhapsody (see Figure 2-17). The anchored element is the component which shall satisfy the Contract.

Figure 2-16: Edit a Contract
Figure 2-17: Link Contract to model element
3 Constituent System Modeling Tools

3.1 Rhapsody

Rhapsody being an off the shelf product is used in both the System of Systems (SoS) level and the constituent level. Commonly, the constituent level would be considered a SysML modeling, while the SoS level that would be the use of UPDM views of models. The tool integrates with the tool-net by sharing models with the semantic mediation platform. The semantic mediation for these models on the platform can be mediated with models in the Modelica tool from Wolfram: System Modeler.

This section describes the interoperability

3.2 Abstraction tools – Statistical Learning

3.2.1 Purpose

The abstraction technique “Statistical learning” generates abstractions of systems – CS or whole SoS – from observations of the system's behaviour. This is particularly useful in case it is not possible to get a system model in another way, for example because the system is too complex to be modelled in detail, or simply because the internal functioning of the system is not known.

Based on the input and output behaviour of the system – given as a data stream – the technique learns the statistical behaviour of these streams, and generates a stochastic model that reproduces this behaviour statistically. The technique has a number of advantages: it is fast, almost automatic, and – through continuous learning – can adapt to changing conditions. Requirements are that the data streams are large enough to gain accurate statistical quantities, are at least stationary for a limited time and could be described by a multi-dimensional markovian system(memory-less).

3.2.2 Tools and environment

The method of “Statistical learning” is available as a FMU that could be integrated into existing models like any other FMU. If this FMU does not require any modifications for the current objective it could be included by using the FMU-Importer available in Rhapsody. Rhapsody (see chapter 2.1) also could be used to define any necessary FMU parameters. The basic “Statistical learning” FMU uses a maximum of up to three correlated input signals, but could be extended or adapted by changing the corresponding MATLAB Simulink project of the FMU. These changes could be exported from MATLAB Simulink by using the DYMOLA Simulink FMU Export (see chapter 4.2) and treated in the same way as any other FMU that is used for
modelling purposes. The “Statistical learning”-FMU could also be used within DESYRE (see chapter 4.4) for simulation.

### 3.3 Modelica w/ System Modeler

Off the shelf tool by Wofram® - System Modeler is a Modelica tool which has been used in the SPRINT [5] project and integrated into the tool-net semantic mediation flow and is now capable to exchange models through that flow with Rhapsody SysML models.

The tool is extended with the tool-net capability and can be obtained for that purpose from the product owners.

The version download and license modifications will be done with Otto Tronarp ([mailto:ottot@wolfram.com](mailto:ottot@wolfram.com))

Any use of this tool must start with contacting this technical contact point.

Initial license purchase can be followed with Daniel Liezrowice - ESL ([mailto:mailto:ottot@wolfram.com](mailto:mailto:ottot@wolfram.com)) .
4 Joint Simulation and Analysis Tools

4.1 Rhapsody SysML FMU exporter

4.1.1 FMU generation on Rhapsody

Rhapsody FMI plugin was developed to export Rhapsody SysML blocks as FMUs. The current plugin version supports export to FMI 1.0 for model-exchange. The plugin uses regular Rhapsody code generation, while in addition FMI wrapper and XML Mode description are generated. The plugin defines the following mapping from SysML to FMI (Table 4-1):

<table>
<thead>
<tr>
<th>SysML element</th>
<th>FMI element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>FMU</td>
</tr>
<tr>
<td>Atomic input flowport</td>
<td>Scalar input discrete variable</td>
</tr>
<tr>
<td>Atomic output flowport</td>
<td>Scalar output discrete variable</td>
</tr>
<tr>
<td>&lt;&lt;FMUParameter&gt;&gt; attribute</td>
<td>Scalar internal parameter variable</td>
</tr>
<tr>
<td>“not annotated” attribute</td>
<td>Scalar Internal discrete variable</td>
</tr>
<tr>
<td>… const</td>
<td>Constant</td>
</tr>
<tr>
<td>Attribute initial values</td>
<td>Start value of scalar variable</td>
</tr>
</tbody>
</table>

Table 4-1: Mapping from SysML to FMI.

Here are the main steps of the export process:

1. XML Model description generation
2. Code generation for SysML block
3. Code generation for FMI wrapper
4. DLL compilation
5. Archiving binaries and model description into FMU file.

That file can then be used by the DESYRE [23] tool as described above.

4.1.1.1 Demonstration scenario

The CAE [38] behavioural model will be used as demonstration scenario. Objective of the demonstration is to show how behavioural models can be exported to FMU and integrated into DESYRE [23] to simulation the behavioural aspects of the overall SoS. The export to an FMU of the UPDM behavioural representation of a system within the SoS is achieved via the Rhapsody plug-in provided by IBM (see 4.1.2 below). Integration into the simulation framework DESYRE occurs according to the system view diagram that specifies how the
different systems interact. DESYRE will provide a Rhapsody plug-in to export the UPDM system view to an intermediate representation that is shared via the tool-net. The DEYSRE dashboard is able to read the intermediate representation of the system view from the tool-net and replicate it into DESYRE. DESYRE also collects the FMU system implementations from the tool-net according to the URI references specified within the system view.

Once the model has been correctly constructed for DESYRE, the user can configure and run simulations of the SoS via the DESYRE dashboard and collect the results of the analysis. The user can also specify performance metrics that provide aggregate results, for a single simulation (level 1 metrics) or multiple simulations (level 2 metrics) within the same analysis session, that allow the user to evaluate trade-offs of alternative model configurations.

### 4.1.2 Rhapsody model export

This capability comes with a plugin which enables SysML projects to be exported in RDF to the tool-net semantic mediation platform. Models can be re-exported to deliver modifications to the model, as well as imported, to consolidate into the model possible changes and contributions from other tools to the same model. The capabilities of the semantic mediation platform are described below (see 5.1), with which this tool operates. When FMU capabilities need to be integrated with the semantic mediation platform to facilitate simulation of components, the association of FMU objects with the appropriate elements of the model are maintained by this plugin capabilities.

#### 4.1.2.1 Demonstration scenario

1. A Rhapsody project is equipped with the sm-dm-2.1 plugin which implements the Rhapsody adapter for integrating with the semantic mediation container. In the left picture, we see how a new project needs to be defined as a SysML project so it can be used by the adapter. On the right side, both old and new projects follow the procedure to add a profile to a model, where the sm-dm-2.1.sbs file is picked from the sm-dm-2.1 profile folder of Rhapsody.

![Figure 4-1: Installing the Rhapsody plugin for semantic mediation export/import SysML models.](image1)

![Figure 4-2: Applying this plugin as a profile to a project.](image2)
2. Once an existing model installs the adapter, the adapter is initialized as can be seen on the log console of Rhapsody, and right clicking on the project, shows adapter commands for exporting and importing to the platform names here “IoSE”, or doing the same to/from a file. The file would be an XML file containing the RDF model being exported. Working with the platform server, the same RDF model is stored in a repository on the platform and it can then be browsed via the web.

![Activate the export command](image1)

**Figure 4-3: Activate the export command**

![Setting up user credentials and target server.](image2)

**Figure 4-4: Setting up user credentials and target server.**

---

### 4.2 Dymola Simulink FMU export

#### 4.2.1 Purpose


#### 4.2.2 Tools and environment

Dymola is an off-the-shelf tool based on the Modelica standard language ([http://www.modelica.org](http://www.modelica.org)). Dymola has an FMU export feature to generate FMI executable models from Modelica models. The dymola distribution also provides a Simulink coder target for FMI code generation that enables Simulink users to export Simulink models as FMI executable models.
Link to Dymola web page (note that there is a Demo version of the tool): http://www.dymola.com

Link to Simulink web page: http://www.mathworks.it/products/simulink/

### 4.3 PLASMA statistical model checking

#### 4.3.1 Purpose

Perform statistical analysis of a model containing contracts, through DESYRE [23], directly activating the PLASMA-LAB [25] analyser for that. PLASMA-LAB is a Statistical Model Checker, i.e. it estimates the model satisfaction for some properties. The properties describe some expected behaviour of the model using the standard temporal logics (Linear Temporal Logic) whose give a high level of expressivity and preciseness. In comparison with traditional Model-Checking, the main advantage is scalability of the techniques: the SoS’s models are generally so big to be analysed with model-checking, whereas since SMC just monitors the model and checks the contracts for a set of random executions and computes a reliable estimation, thanks to the mathematical results from the statistics area.

#### 4.3.2 Tools and environment

PLASMA-LAB is distributed as SMC library or a standalone tool. In the DANES settings, it is used as library and plug into DESYRE to extend the simulation toolset with SMC. From the DANSE tool-net point of view, PLASMA-LAB is thus not visible as a standalone module but as a plugin of DESYRE. The PLASMA-LAB workflow is then a subset of the DESYRE workflow presented in Fehler! Verweisquelle konnte nicht gefunden werden. The user is able to pilot PLASMA-LAB through the DESYRE dashboard that has an extension dedicated to the SMC functionalities. The SMC results are returned to DESYRE that disseminates them over the tool-net. Figure 4-5 gives more details about the PLASMA-LAB workflow especially with DESYRE.
As already explained in Subsection 4.4.2, the DESYRE collects the compiled model from Rhapsody. Similarly, it also gets the contracts attached to this model. These contracts defined using the Goal Contract Specification Language that completes the UPDM modelling by attaching a formal specification of some goals. This language is more readable and thus easier to understand than LTL [26]. It is closer to the handwritten specification but with a formal semantics whereas LTL is a very low level language and LTL is generally understood by experts only. Each GCSL [24] goal is translated by an external compiler invoked by DESYRE to produce an equivalent LTL formula, the input specification language of PLASMA-LAB. Next, the user can start a SMC [27] session, to start PLASMA-LAB that starts the analysis. This analysis is based on simulations that are provided by DESYRE each time PLASMA-LAB requires new simulation: During the SMC session, DESYRE becomes temporarily the plugin of PLASMA-LAB. When the session finished, PLASMA-LAB returns the results collected by DESYRE.

### 4.3.3 Demonstration scenario

As for the DESYRE demonstration, the CAE incubator will be used as demonstration scenario. The CAE incubator provides a UPDML model designed in Rhapsody plus a set of contracts in GCSL. The satisfaction of these contracts can be estimated over the model: from the results the user may take some decisions to modify the architecture of the CAE in order to enhance the satisfiability of the contracts.
4.4 DESYRE joint simulation

4.4.1 Purpose
In the DANSE tool-chain DESYRE [23] provides the framework for joint simulation (i.e. simulation of heterogeneous models), performance evaluation and property-satisfaction analysis of Systems of Systems (SoS); it also interfaces to PLASMA-LAB [25] verification engine providing simulation traces used to perform statistical model checking.

4.4.2 Tools and environment
Inputs to DESYRE are a structural model representing the SoS architecture, i.e. how Constituent Systems (CS) are interconnected, and a model describing each CS. Models for Constituent Systems can be developed using different tools (e.g. Modelica [33], Jmodelica [35], Dymola [34], Rhapsody [37] and Simulink/StateFlow [36]) as long as the tool has the capability of exchanging the model in compliance with FMI 1.0 standard [32]. The SoS structure is instead provided as a UPDM [8] diagram using a set of DANSE extensions to the UPDM profile, which can be loaded into the IBM Rhapsody tool.

Both the SoS structure and the constituent system models (exported as FMUs) are published on the DANSE tool-net, operation that makes them available for use by the other tools interconnected by the tool-net. Once the SoS model is complete, the designer can annotate it with a set of auxiliary information, such as goals, contracts, stochastic parameters, metrics and so on. This can be done using the set of stereotypes provided by the DANSE profile for UPDM. Completed the specification and annotation of the SoS model, the designer will be able to use IBM architect to perform some architecture optimization of the SoS and GROOVE [22] to generate possible life-cycle evolutions of the SoS during its life-cycle. For each candidate SoS model resulting from both the architecture optimization and the life-cycle evolution evaluation phases, simulation can be used to provide performance analyses.

![Diagram: DANSE System of System Design & Simulation Workflow]

Figure 4-6: DANSE System of System Design & Simulation Workflow
Once the simulator and the analysis tools have been configured, the simulation can be run.

4.4.3 Demonstration scenario

The CAE [38] model and the industrial use cases provided by the DANSE partners will be used as demonstration scenarios. The following pictures and screenshots represent a quick overview of the tool position into the DANSE tool-chain and a flavour of its usage; the demonstration aims at showing the import from toolnet, the analyses configuration editors, the analyses capabilities and how to interpret the results.

![DESYRE Import wizard](image)

**Figure 4-7: DESYRE Import wizard**

The figure above shows the DESYRE’s import wizard through which the user imports the intermediate representation of the Rhapsody SoS model and the FMUs. Both the intermediate model, generated by the DANSEExporter Rhapsody plugin, and the FMUs can be imported from the user’s file system and from the Tool-Net.
DESYRE provides a configuration pane for each of the analyses it supports: joint simulation, statistical model checking (together with PLASMA-LAB), contract-based run-time verification, GCSL contract analysis and fault-based test generation. As an example, the picture above shows the joint simulation configuration.

### 4.5 TestCast test generator

Test case generation based on the TTCN-3 standard and MBT. This is an off the shelf tool by Elvior®, which works with Rhapsody state-charts for the description of blocks behavior and based on that description, generates test cases to test an implementation of that block. The tool is also integrated with the tool-net and works with Rhapsody models shared over the semantic mediation tool-net platform. The tool is not part of the DANSE foreground and needs a user licence from the vendor. The integration capability has been developed in the SPRINT [5] project.

- Price and licensing terms: [http://www.elvior.com/testcast/licensing](http://www.elvior.com/testcast/licensing)
  for DANSE project there are 5% discount from list price.


- Contact: Andrus Lehtmets [andrus.lehtmets@elvior.ee](mailto:andrus.lehtmets@elvior.ee)
4.6 Contract-based Run-time Verification

4.6.1 Purpose

Within the Synthesis for Diagnosis and Prognosis task, run-time verification of executable specifications plays a central role. Purpose of this tool, integrated in the DESYRE simulation framework [23], is to provide simulation-based verification means for goals and contracts specified through the GCSL language.

4.6.2 Tools and environment

The user specifies the goals and contracts using GCSL statements within the System View of the SoS UPDM model. The SoS architecture specified in the System View is imported from the tool-net into DESYRE together with the FMU executable models of the different constituent systems in order to build an SoS executable model. The run-time verification tool automatically generates executable models of GCSL goals and contracts, called monitors, that are linked with the SoS executable model. Contract monitors produce a Boolean value that become false when the contract is violated. Goals monitors produce quantitative results instead that can be viewed and post-processed by the user to assess the performance of the SoS and identify conflicting goals.

4.6.3 Demonstration scenario

The CAE [38] model and the industrial use cases provided by the DANSE partners will be used as demonstration scenario. The satisfaction of GCSL contracts can be observed by simulation as the output of the contract monitors.

![Diagram](image)

**Figure 4-9: GCSL in DANSE**
4.7 GCSL Contracts Analysis

4.7.1 Purpose

The GCSL Contract Analysis technique aims at providing an early validation tool for GCSL Contracts at SoS level and at Constituent Systems (CS) level. GCSL Contracts constrain the values over time of the inputs and outputs of a component; thus, they can be seen as behavioural specifications at a high level of abstraction. This analysis has the purpose (1) of checking whether the requirements expressed by the GCSL contracts are consistent or if they are in conflict (Compatibility and Consistency) and (2) of checking whether the allocation of functionalities to single components allows to meet the top-level SoS requirements (Dominance). GCSL Contract Analysis is a light technique that can help to spot problems at an early design stage, where implementation or detailed behaviour is not fully developed yet.

4.7.2 Tools and Environment

The tool provided in this Prototype covers Dominance checking for the fragment of GCSL including time and OCL but not probability. The tool is integrated within DESYRE [23] and has a use flow following that of the other analyses provided within DESYRE: (1) import of the architecture (including the GCSL Contracts), (2) user set up of the analysis parameters, (3) report of the analysis results, providing evidence of errors in the requirements by computing explicit conditions under which the requirements are violated.

4.7.3 Demonstration scenario

The first demonstration scenario for the GCSL Contracts Analysis tool is the IWTS Test Case from IAI, where GCSL Contracts (used to model event causality and timing requirements) are added to Constituent Systems as well as to the SoS, to assess that timing allocations are correct. This is formulated in terms of Dominance between Constituent Systems Contracts and SoS Contracts. Further experiments will be performed on the reference test case CAE WayForward, again in terms of Dominance checking,
4.8 Fault-based Test Generation

4.8.1 Purpose

Fault-based test generation aims at increasing the designer confidence about the robustness of the SoS to the occurrence of faults. Faults occurring to (1) Constituent Systems, or to (2) Constituent Systems’ interconnections can propagate through the SoS architecture causing the SoS to behave unexpectedly (unwanted SoS behaviour). The objective of fault-based test generation is to identify a set of input traces to be used in the simulation phase to (1) test the occurrence of unwanted SoS behaviours operating in presence of faults and (2) to identify corner cases exposing modelling errors.

4.8.2 Tools and Environment

Inputs to the fault-based test generation tool are: (1) the SoS structure specified as an UPDM sv-1 diagram, (2) the nominal (fault-free) Constituent Systems’ behaviour specified either as UPDM statecharts or as a set of GCeSL contracts, (3) the set of faults specified either as UPDM statecharts or as a set of GCeSL contracts and (4) the unwanted SoS behaviour(s) monitor specified either as UPDM statecharts or as a set of GCeSL contracts. The fault-based test generation tool is integrated by DESYRE; the aforementioned system structure, UPDM statecharts and GCeSL contracts are imported from the tool-net.
For each unwanted SoS behaviour the tool generates a test suite, which is the set of simulation traces. Such traces are used to exercise the SoS fault detection mechanisms to verify (by simulation) the SoS tolerance to the modeled faults.

### 4.8.3 Demonstration scenario

A simplified version of the CAE [38] use case, further tailored to support fault models and SoS unwanted behaviour model(s) will be used as demonstration scenario. The SoS tolerance to the modelled faults is verified through simulation by observing the activation of the unwanted SoS behaviour monitors.

![Figure 4-11: Fault-based test generation within the DANSE tool-net.](image)

### 4.9 Timing analysis

#### 4.9.1 Purpose

The timing analysis tool (realtimeAnalyzer) considers parts of the SoS where timing requirements are annotated to constituent systems and their functions. This tool performs the validation of these requirements. In case of a violation of a time bound by some constituent systems a counter example leading to the critical state is generated.

The main features are the following:
- Analysis of local and global timing requirements for set of constituent systems with sets of operational nodes
- Timing analysis including interfering processes (e.g. operational activities). Hereby, interferences may occur, when processes have different priority levels and may pre-empt each other

In order to alleviate the problem of state space explosion due to state unfolding, the approach constructs the state space of a constituent system in a compositional manner. To this end, abstraction and composition operations allowing constructing the state spaces separately for individual constituent systems were implemented. The state spaces of those CS-s needed for dependent CS are appropriately abstracted to keep only the necessary parts. In the case where the state space calculation of depends on multiple other resources, the corresponding (abstracted) state spaces are composed.

### 4.9.2 Tool & Environment

The input of the timing analysis tool is a UPDM model enriched by timing annotations from the MARTE profile and GSCL constraints for local and global timing requirements. The tool is then called with the model to be analysed. The result of the tool are either the response times of all functions of the constituent systems in case all timing constraints are fulfilled, or a counter-example. The counter-example includes the state sequence together with the corresponding event sequence, which leads to a state where an end-to-end latency or the deadline of a function is violated.
5 Semantic Mediation and Integration

5.1 Tool-Net Semantic Mediation Container (SMC) Platform

This is the semantic mediation platform – which we call the IBM Semantic Mediation Container – SMC.

5.1.1 Purpose

Semantic mediation platform on Jazz/DM serving mediation of RDF models between Rhapsody and a common SysML ontology. This SysML ontology is common to a Modelica tool that is not included in the prototype, not being part of the DANSE consortium, although the mediation to an RDF compatible with this tool is included on the platform. A separate mediation for UPDM/NAF is described in section 5.3 below, all sharing the same interoperability principle and architecture that are described in the D.8.2.3 deliverable [31]. Client tools of the platform includes already several tools, and an SDK for developing new clients is provided as one of the DANSE technologies for this purpose.

5.1.2 Tools and environment

In a nutshell, the semantic mediation platform is a Jazz/DM plugin on which a network of mediation can be configured where RDF models are transformed according to ontologies governing the contents of model repositories. Models are originated from participating tools as exported models from the tools. Tools can also import models from the platform.

In this scenario, tools act as web clients, and the platform is a web server. The protocol is RESTful protocol where tools can POST models to the server, which in turn triggers a chain of mediations which ends when all connected repositories in the network are updated.

A tool can also GET a model, which is an “import” activity. The tool and the server need to be smart enough to handle properly updates of existing models when posts (exports) and gets (imports) are repeated.

5.1.2.1 Jazz/DM

This is the server platform on which the “Semantic Mediation Container” is a plugin. The container can host pluggable mediators. In this particular scenario we use several mediators. In particular a mediator developed in the SPRINT project is exploited here, which is capable to perform transformation among models. This mediator works according to rules which are coded in OWL ontology that specifies equivalence and conditions for such equivalences among classes and properties from the relevant ontologies – those associated with repositories on the server that are configured for a mediation link.

Other mediators are the “Null” mediator and the “Extractor” mediator. The first is a simple copy mediator which does not change the model structure. The latter is a mediator which interprets a model according to a tree structure so that proper sub-models can be cut out of a larger model. The extractor mediator works according to rules coded in OWL ontology.
Powerful mediators are provided by IBM – the Haifa Mediator, and SODUIS (see chapter 5.3), which mediate models in a generic way. The IBM mediator uses ontological rules, while SODUIS uses the MDWorkbench rules editor.

### 5.1.2.2 Rhapsody

This modeling tool is enhanced with a plugin (also termed “profile” in the Rhapsody terminology) which implements the integration pattern A (see Chapter 6 in [21]). This plugin enhances the GUI with export and import commands to the “IoSE” – which is the server on the internet.

### 5.1.3 Demonstration scenario

1. A Rhapsody project is loaded with the SMC plugin which is part of the DANSE profiles (modeling) extensions (see chapter 2.6). This profile implements the Rhapsody adapter for integrating with the semantic mediation container. In the left picture, we see how a new project needs to be defined as a SysML project so it can be used by the adapter. On the right side, both old and new projects follow the procedure to add a profile to a model, where the SMC.sbs file is picked from the SMC profile folder of Rhapsody. Yet, with the DANSE profile (Chapter 2.6), this profile is already included.
Figure 5-1: Installing the Rhapsody plugin for semantic mediation export/import SysML models as part of the DANSE profiles (Chapter 2.6). The console shows the version and initialization of this profile plugin.

Figure 5-2: Version identification of components of the profiles including the SMC plugin profile.
2. Once an existing model installs the adapter, the adapter is initialized as can be seen on the log console of Rhapsody, and right clicking on the project, shows adapter commands for exporting and importing to the platform names here “SMC”, or doing the same to/from a file. The file would be an XML file containing the RDF model being exported. Working with the platform server, the same RDF model is stored in a repository on the platform and it can then be browsed via the web.

Figure 5-3: Activate the export command

Figure 5-4: Setting up user credentials and target server.

3. Exporting to the server.

Figure 5-5: Successful export done.

4. Import to a new project
## Figure 5-6: Create a new UPDM project and apply to it the DANSE plugin.

![New Project](image)

**Figure 5-6: Create a new UPDM project and apply to it the DANSE plugin.**

## Figure 5-7: Apply the SMC import command

![SMC Import](image)

**Figure 5-7: Apply the SMC import command**

## Figure 5-8: Enter user credentials, source server IP and URI of an element to import.

![User Credentials](image)

**Figure 5-8: Enter user credentials, source server IP and URI of an element to import.**

## Figure 5-9: Populate a diagram from the created model for this import.

![Populate Diagram](image)

**Figure 5-9: Populate a diagram from the created model for this import.**
5.1.4 Enhancement in Prototype III

UPDM capabilities have been added, extending the SysML ontology of the RDF model with UPDM. That creates the opportunity to mediate between UPDM and SysML. An important application of this capability is in bridging the “Concise” technology which is developed to handle SysML models only, and apply it also to UPDM models. The flow is to export UPDM models, mediate them to SysML, apply the “Concise” processing, resulting with an enhanced SysML model which can be used as is by the engineer to evaluate the optimization results, or mediate back to UPDM and load the results into the UPDM model and then view and evaluate the results.

5.2 Protégé ontology editor

Note: No significant changes in prototype III.

Protégé is an open source public domain OWL [15] ontology editor. The tool can be extended with plugins one of which has been developed for the SMC.

Once Protégé is downloaded and installed as in the next figure, the plugin can be installed in the Protégé installation location, under the plugin/ folder.

![Protégé Desktop 4.3](image)

**Figure 5-11:** Installing Protege 4.3.
Running Protégé will detect the SMC plugin and bring up the following interacting panel through which ontologies of the SMC can be captured for modifications or just inspection in the Protégé user presentation desktop.

![Control panel of the SMC plugin to Protege](image)

**Figure 5-12:** Control panel of the SMC plugin to Protege

The Protégé editing desktop will reflect OWL ontologies of SMC as in the next figure:
Figure 5-13: Protege editing session with an ontology from SMC.
5.3 MDWorkbench mediation rules editor

MDWorkbench is a SODIUS tool by which rules for mediating among models can be coded. The environment in which the MDWorkbench transformations work is the SMC, and the work flow in developing such mediators is explained in section 2.1.3.

5.3.1 Purpose

MDWorkbench is a framework based on EMF for:

- Defining Model Driven Architectures
- Using and creating a large set of Accessors to third party tools
- Models can imported/export through XMI files or specific accessors
- Models can be transformed to other metamodels by Rulesets (MQL + Java)
- Models can be transformed to code or other kind of text files through text templates (TGT + Java)

MDWorkbench for Danse provides:

- A generic interpreter of MDW Rulesets
- An exporter of Rules definition to Danse Toolnet
- An exporter of Ecore Metamodels to OWL Ontologies

5.3.2 Usage

MDWorkbench is bundled with a complete documentation available directly into the tool, describing all concepts relatives to metamodel creation, ruleset definition, and accessor extension.

Figure 5-14: Example of MDWorkbench documentation
All Danse concepts as developing a semantic mediator rule or adapting a metamodel to the ToolNet, are
detailed in a tutorial describing all development steps for bidirectional semantic mediator.
This tutorial contains both documentation and source code.

![Semantic Mediator Tutorial Image](image)

**Figure 5-15**: A page from MDWorkbench semantic mediator tutorial

### 5.4 SMC client SDK

**Note**: No further changes in Prototype III.

The developer role of tool-net users is the one of tool vendors who wish to integrate the tool with the tool-net
platform – SMC. The SMC client SDK is a java JAR which includes an implementation of the SMC protocol
and a user GUI for managing the interaction of users driving the tool for export and import of tool models.

The SDK is used with some client program which is a model development tool. The next screen shot shows
an example client which can deliver simple pre-built models in RDF to the SMC:
Figure 5-16: Example SMC client to demonstrate the SDK.

Being an SDK, this tool comes with a detailed javadoc documentation of the API.

Figure 5-17: Javadoc documenting the SMC client API.
5.5 FMU Importer Client

Note: This is a new technology item in prototype III.

Using the SDK (chapter 5.4), an FMU importer is developed for the 3rd prototype and is still in progress when this report is written. This client takes FMU objects in which an XMI structure describes the component interface according to a specific XSM schema, the tool converts that XMI into an RDF, using an FMU ontology in OWL. Mediation is performed on the SMC tool-net to mediate such models into SysML which can then be used by other mediation compliant tools such as Rhapsody, Modelica, UPDM and System Architect, DESYRE [23], and any future tools that can be added to the tool-net.

5.6 Model Instances Generator Client

Note: This is a new technology item in prototype III.

Using the SDK (chapter 5.4), this client of the tool-net uses the tagging language of Concise to process a model and generate numerous instances which can then be used back in the “instances” model to perform various test scenarios.

The method is to work in Rhapsody to tag model classes and blocks with proper stereotypes from the Concise profile. The tagged model is exported to the tool-net (SMC), from which it can be imported into the model instantiation tool, processed and generating a corresponding Excel data sheet in which users can fill up instantiation parameters for the model classes and blocks. The Excel data sheets are then used to produce instances as a model in the RDF format that can be exported back into the tool-net and used in any tool that can import the model or a semantically-mediated version of that model.

This technology item is still in development at the time of writing this report, and is intended as part of the 3rd DANSE prototype.
The 3rd DANSE prototype is presented here as yet another milestone, with some progress over the 2nd prototype which consists mostly of strengthening the tools based on initial usage by the industrial partners, and reflection on their input and demands. A few additional technologies were added, and there are still more work in progress that cannot make it to the delivery time of this document.

Integration of tools with the tool-net and the semantic-mediation container have also progressed and will continue after the delivery of this document.

DANSE will make further progress to meet evolving users’ demands and as the methodology is also evolving to reflect on the experienced gained. It becomes clearer at this point which tools are used by the partners, certainly not all the tools and in different levels by the various users.

A final picture of the tools usage and value to the partners will only become clear at the end of the project.
7 Abbreviations and Definitions

**ASDI** Aircraft Situation Display to Industry

**Application** A software program that provides added value on top of tools by applying functions that have not been addressed by individual tools and that are possible due to the integration of data from multiple tools.

Applications that add a new value to the data in the TOOLNET repository are referred to as AVAs – Added Value Applications.

**CAE** Concept Alignment Example

**CS** Constituent System.

**Data scoping** When shared, we distinguish several levels of scoping in data, such as private, internal, and public. Reasons for data scoping may be protection of rights as well as technical such as proprietary information. There may be more categories, yet presently we can discuss only these three levels:

- **Private** Data that is located on and managed only by the tool. It may be available to applications by accessing the tool via some standard API (such as OSLC).
- **Internal** Data that is shared and may be enriched to match a certain level of compatibility with the information bus, but is not shared with other partners.
- **Public** Data that is shared with other partners.

**Data sharing** For specific tools, the data for a certain engineered system can be shared with other tools and applications. When data is shared, it is "exported" to the TOOLNET since we assume the only way to share the data is via the information bus implemented by the TOOLNET.

**DEE** The DANSE Engineering Environment consisting of the following

- **Tool Net** The Tools Interoperability facilities and the Integration platform.
- **SSI** The Semantic Services Integration layer of the ToolNet platform

**DODAF** Department of Defence Architectural Framework
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM</td>
<td>Design Management application of Jazz. Used to define modeling domains and provide visualization over the web of corresponding modeling data.</td>
</tr>
<tr>
<td>DTK</td>
<td>Design management ToolKit. Used for developing new ontology meta-models (domains) in the DM</td>
</tr>
<tr>
<td>Elements</td>
<td>Nodes constituting the model data of a project. The model also consists of relations between these elements.</td>
</tr>
<tr>
<td>Enrichment</td>
<td>Tool data when exposed and exported to the TOOLNET for sharing must be enriched to integrate with data from other tools serving the same developed system. Enrichment depends on the applications intended to use that data; as new applications are developed and enhanced, the requirements from the enrichment function may change.</td>
</tr>
<tr>
<td>FMI</td>
<td>Functional Mockup Interface</td>
</tr>
<tr>
<td>FMU</td>
<td>Functional Mockup Unit</td>
</tr>
<tr>
<td>GCSL</td>
<td>Goal Contract Specification Language. Language designed to extend the UPDM profiles by attaching some local of global goals to the SoS constituents. (See [40] for more details)</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information System</td>
</tr>
<tr>
<td>GXL</td>
<td>Graph eXchange Language - an xml-scheme which is used by GROOVE.</td>
</tr>
<tr>
<td>JIA</td>
<td>Jazz Integration Architecture lays out the architecture for integrating services and application within the Jazz framework.</td>
</tr>
<tr>
<td>JTS</td>
<td>Jazz Team Server is the core services provider of the Jazz platform</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol is the communication protocol over the Internet which is used to connect Web clients (browsers and applications) and servers.</td>
</tr>
<tr>
<td>HRC</td>
<td>Heterogeneous Rich Components</td>
</tr>
</tbody>
</table>
**Links**

Relations between element nodes in a model are known as links. There are two kinds of links:

- **Intra-links**: Internal relations between elements of a model emanating from a single tool instance. These are natural links defined in that tool or such that are introduced or modified during the enrichment.

- **Inter-links**: Relations between elements originating in models from different tools or from different projects. These relations can only result from enrichment, either during data exportation (publishing) or during enrichments taking place in the TOOLNET, using automatic or manual tools.

**LTE**

Long-Term Evolution, marketed as 4G LTE, is a standard for wireless communication of high-speed data for mobile phones and data terminals.

**LTL**

tbd

**MODAF**

Ministry of Defense Architectural Framework

**NAF**

NATO Architectural Framework

**OAUTH**

An Authentication protocol that is used by Jazz to provide secured interaction over the internet of users and the Jazz platforms.

**Open Services for Lifecycle Collaboration**

Open Services for Lifecycle Collaboration (also known as OSLC or Open Services) is a community and set of specifications for Linked Lifecycle Data. The community’s goal is to help product and software delivery teams by making it easier to use lifecycle tools in combination.

See: [http://open-services.net/html/Home.html](http://open-services.net/html/Home.html)

**OWA**

Open World Assumption

**OSLC**

See: Open Services for Lifecycle Collaboration

**OSLC-AM**

The Architecture domain of the OSLC specifications.

**OSLC-CM**

The Change Management domain of the OSLC specifications.

**PD**

Physical Device

**OSLC-RM**

The Requirement Management domain of the OSLC specifications.
Project
A component that is engineered collectively over a set of tools, and which is subject to processing by some of the applications. It must be clearly identified across TOOLNET and all the relevant tools.

Project Publishing
Exporting project-related data stored in a certain tool into the TOOLNET. This mechanism also includes an enrichment function.

Resource
Identified element or relation in any model data that is stored in the TOOLNET and which can identify back the original element in the originating tool. Some of the resources are generated by the TOOLNET – such as enriched data: elements and links. A resource has a single owner.

Resource Description Framework
It’s a family of W3C specifications for conceptual description or modelling of information that is implemented in web resources. (See: http://www.w3.org/TR/rdf-primer/)

RDF
See: Resource Description Framework

RTP
Reference Technology Platform (of CESAR)

SDK
Software Development Kit. Related to the development environment of services over Jazz/DM.

Semantic Mediation
The transformation of model data between models according to the semantics of the modeling languages, and in an incomplete way according with the Open World Assumption (OWA).

SM Container
A Jazz/DM plugin which executes mediation flow paths through mediators to carry out the semantic mediation task of model collaboration in the DANSE tools-net eco-system.

SMC
Semantic Mediation Container

SMC
Statistical Model Checking

SPARQL
SPARQL (SPARQL Protocol And RDF Query Language) is a query language for RDF. http://www.w3.org/TR/rdf-sparql-query/

SysML
Systems Markup Language

Tool
A software program that models some aspects of a product’s design. Tools have internal models of the design and can serve as part of a group of tools that together serve the full engineering process. However, used by itself, a tool is also an independent program with its own data repository and management and usability functions that allow users to work with it totally independent of other tools. A tool generally is said to hold some information about the engineered system.
**Tool data**  A model based on a well-defined meta-model that defines a certain aspect of an engineered system. For instance, the aspect can be the functional requirements of the product, and the model must be detailed enough so that each requirement can be assigned to a specific component of the system. Meta-models can also associate additional information such as the relations (structural, logical, or geometrical) between the components.

**Tools/Data isolation**  A mechanism that implements a set of rules for the access permissions by applications to certain portions of the tools' public data. Note that while access control may not be needed to functionally implement TOOLNET, it is a mandatory property of an TOOLNET that can be used commercially to collaborate between distinct private vendors.

**UML**  Unified Markup Language
## 8 References

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[18] D3.1</td>
<td>D3.1 &quot; Project Requirements”</td>
</tr>
<tr>
<td>Page</td>
<td>Reference</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>82</td>
<td>[22] GROOVE</td>
</tr>
<tr>
<td>82</td>
<td>[26] LTL</td>
</tr>
<tr>
<td>82</td>
<td>[27] SMC</td>
</tr>
<tr>
<td>82</td>
<td>[28] DM</td>
</tr>
<tr>
<td>82</td>
<td>[29] SMa</td>
</tr>
<tr>
<td>82</td>
<td>[31] D8.2.3</td>
</tr>
<tr>
<td>82</td>
<td>[33] Modelica</td>
</tr>
<tr>
<td>82</td>
<td>[34] Dymola</td>
</tr>
<tr>
<td>82</td>
<td>[38] CAE</td>
</tr>
<tr>
<td>82</td>
<td>[40] D6.5.1</td>
</tr>
<tr>
<td>82</td>
<td>[41] CPLEX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[42]</td>
<td>Prototype-I</td>
<td>DANSE deliverable D8.3 – Prototype I.</td>
</tr>
<tr>
<td>[43]</td>
<td>D3.6</td>
<td>DANSE deliverable D3.6 – Validation Report - Prototype Iteration I</td>
</tr>
<tr>
<td>[47]</td>
<td>D3.7</td>
<td>DANSE deliverable D3.7 – Validation Report – Prototype II</td>
</tr>
<tr>
<td>[48]</td>
<td>D8.4</td>
<td>DANSE deliverable D8.5 – Prototype II.</td>
</tr>
</tbody>
</table>