Designing for adaptability and evolution in system of systems engineering

DANSE Prototype - Final Report

D_8.8

<table>
<thead>
<tr>
<th>Deliverable Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nature</strong></td>
</tr>
<tr>
<td><strong>Project</strong></td>
</tr>
<tr>
<td><strong>Deliverable ID</strong></td>
</tr>
<tr>
<td><strong>Status</strong></td>
</tr>
<tr>
<td><strong>Contact Person</strong></td>
</tr>
<tr>
<td><strong>Phone</strong></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>Tayfun Gezgin</td>
<td>OFFIS</td>
<td><a href="mailto:Tayfun.Gezgin@offis.de">Tayfun.Gezgin@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>Benoît 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>2014-12-21</td>
<td>Initialized from D8.5</td>
<td>All</td>
</tr>
<tr>
<td>0.2</td>
<td>2015-01-28</td>
<td>Updates</td>
<td>4.1, 4.3, 4.5</td>
</tr>
<tr>
<td>0.3</td>
<td>2015-01-29</td>
<td>Updates</td>
<td>1.*, 2.1, 2.2, 2.4, 3.1, 3.2, 3.3, 3.4, 4.9, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 6</td>
</tr>
<tr>
<td>1.0</td>
<td>2015-01-29</td>
<td>Finished, but not final. Need input from LU, OFFIS, SODIUS, ALES.</td>
<td></td>
</tr>
</tbody>
</table>

<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>2015-02-15</td>
<td>2 of 115</td>
</tr>
</tbody>
</table>
1 OVERVIEW, PURPOSE AND SCOPE .............................................................................................................. 11
  1.1 THE TECHNOLOGIES WIKI PAGE .............................................................................................................. 14
  1.2 SOLUTIONS WIKI PAGE ................................................................................................................................. 19
  1.3 TRACEABILITY OF REQUIREMENTS (WP3) AND TOOLS .................................................................................. 22

2 SOS MODELING TOOLS ...................................................................................................................................... 25
  2.1 RHAPSODY ........................................................................................................................................................ 25
    2.1.1 Overview ....................................................................................................................................................... 25
    2.1.2 Sharing UPDM models in the tool-net ............................................................................................................... 25
    2.1.3 Tools and environment ....................................................................................................................................... 26
    2.1.4 Recent Improvements and new features ............................................................................................................. 27
    2.1.5 Demonstration scenario .................................................................................................................................... 28
    2.1.6 Conclusions ...................................................................................................................................................... 30
    2.1.7 Lessons Learnt ............................................................................................................................................... 31
    2.1.8 Future Plans ..................................................................................................................................................... 31
  2.2 SYSTEM ARCHITECT AND ENTERPRISE ARCHITECT ......................................................................................... 31
  2.3 ARCHITECTURE PATTERNS ................................................................................................................................ 31
    2.3.1 Purpose ........................................................................................................................................................... 31
    2.3.2 Tools and environment ...................................................................................................................................... 35
    2.3.3 DANSE SoS Architecture Patterns Library ......................................................................................................... 37
    2.3.4 Demonstration scenario .................................................................................................................................... 39
    2.3.5 Conclusions ...................................................................................................................................................... 41
    2.3.6 Lessons Learnt ............................................................................................................................................... 42
    2.3.7 Future Developments ....................................................................................................................................... 43
  2.4 ARCHITECTURE OPTIMIZATION WORKBENCH .............................................................................................. 43
    2.4.1 Purpose ........................................................................................................................................................... 43
    2.4.2 Tools and environment ...................................................................................................................................... 43
    2.4.3 Demonstration scenario .................................................................................................................................... 45
    2.4.4 Conclusions ...................................................................................................................................................... 51
    2.4.5 Lessons Learnt ............................................................................................................................................... 51
2.4.6 Future Plans .......................................................................................................................... 52
2.5 ARCHITECTURE GENERATION ............................................................................................... 52
  2.5.1 Purpose ................................................................................................................................. 52
  2.5.2 Tools and environment .......................................................................................................... 52
  2.5.3 Demonstration Scenario ...................................................................................................... 52
  2.5.4 Conclusion ............................................................................................................................ 55
  2.5.5 Lessons Learnt ...................................................................................................................... 55
  2.5.6 Future Work .......................................................................................................................... 55
2.6 DANSE MODELING EXTENSION PROFILES .......................................................................... 55
  2.6.1 Conclusions .......................................................................................................................... 56
  2.6.2 Lessons Learnt ...................................................................................................................... 56
  2.6.3 Future Plans .......................................................................................................................... 56
2.7 GCSL EDITOR ............................................................................................................................ 56
  2.7.1 Purpose .................................................................................................................................. 56
  2.7.2 Tools and Environment ......................................................................................................... 57
  2.7.3 Demonstration Scenario ...................................................................................................... 57
  2.7.4 Conclusion ............................................................................................................................ 60
  2.7.5 Lessons Learnt ...................................................................................................................... 60
  2.7.6 Future Work .......................................................................................................................... 60
3 CONSTITUENT SYSTEM MODELING TOOLS ............................................................................ 61
  3.1 RHAPSODY ............................................................................................................................ 61
    3.1.1 Conclusions ......................................................................................................................... 61
    3.1.2 Lessons Learnt ...................................................................................................................... 61
    3.1.3 Future plans ......................................................................................................................... 61
  3.2 ABSTRACTION TOOLS – STATISTICAL LEARNING ............................................................... 61
    3.2.1 Purpose .................................................................................................................................. 61
    3.2.2 Tools and environment ......................................................................................................... 62
    3.2.3 Demonstration Scenario ...................................................................................................... 62
    3.2.4 Conclusions .......................................................................................................................... 64
    3.2.5 Lessons learnt ....................................................................................................................... 64
    3.2.6 Future plans: ......................................................................................................................... 65
  3.3 MODELICA W/ SYSTEM MODELER ......................................................................................... 65
    3.3.1 Conclusions ............................................................................................................................ 65
    3.3.2 Lessons Learnt ...................................................................................................................... 65
    3.3.3 Future Plans .......................................................................................................................... 65
4 JOINT SIMULATION AND ANALYSIS TOOLS ............................................................................ 66

Version  Status   Date       Page
1.0      Final    2015-02-15  4 of 115
<table>
<thead>
<tr>
<th>4.1</th>
<th>Rhapsody SysML FMU Export</th>
<th>66</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.1</td>
<td>FMU generation on Rhapsody</td>
<td>66</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Rhapsody model export</td>
<td>67</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Conclusions</td>
<td>68</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Lessons Learnt</td>
<td>69</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Future Plans</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>Dymola Simulink FMU Export</td>
<td>69</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Purpose</td>
<td>69</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Tools and environment</td>
<td>69</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Conclusions and Lessons Learnt</td>
<td>70</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Future Plans</td>
<td>70</td>
</tr>
<tr>
<td>4.3</td>
<td>Plasma Statistical Model Checking</td>
<td>70</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Purpose</td>
<td>70</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Tools and environment</td>
<td>70</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Demonstration scenario</td>
<td>71</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Conclusions</td>
<td>72</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Lessons Learnt</td>
<td>72</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Future Plans</td>
<td>72</td>
</tr>
<tr>
<td>4.4</td>
<td>Desyre Joint Simulation</td>
<td>72</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Purpose</td>
<td>72</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Purpose</td>
<td>73</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Tools and environment</td>
<td>73</td>
</tr>
<tr>
<td>4.4.4</td>
<td>Demonstration scenario</td>
<td>74</td>
</tr>
<tr>
<td>4.4.5</td>
<td>Conclusions</td>
<td>75</td>
</tr>
<tr>
<td>4.4.6</td>
<td>Lessons Learnt</td>
<td>75</td>
</tr>
<tr>
<td>4.4.7</td>
<td>Future Plans</td>
<td>76</td>
</tr>
<tr>
<td>4.5</td>
<td>Testcast Test Generator</td>
<td>76</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Conclusions</td>
<td>76</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Lessons Learnt</td>
<td>76</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Future plans</td>
<td>76</td>
</tr>
<tr>
<td>4.6</td>
<td>Contract-based Run-time Verification</td>
<td>77</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Purpose</td>
<td>77</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Tools and environment</td>
<td>77</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Demonstration scenario</td>
<td>77</td>
</tr>
<tr>
<td>4.6.4</td>
<td>Conclusions</td>
<td>78</td>
</tr>
<tr>
<td>4.6.5</td>
<td>Lessons Learnt</td>
<td>79</td>
</tr>
<tr>
<td>4.6.6</td>
<td>Future Plans</td>
<td>79</td>
</tr>
<tr>
<td>4.7</td>
<td>Gcsl Contracts Analysis</td>
<td>79</td>
</tr>
</tbody>
</table>

<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>2015-02-15</td>
<td>5 of 115</td>
</tr>
</tbody>
</table>
Figures

Figure 1-1: Tools net for the DANSE prototype II ................................................................. 12
Figure 1-2: Tools connectivity for Prototype III ................................................................. 12
Figure 1-3: The final DANSE tool-net prototype ............................................................... 13
Figure 1-4: Diagraming the concept of model sharing over the Tool-net open "bus" .......... 14
Figure 1-5: One of the work-flows cases discussed in the solutions wiki page .............. 20
Figure 1-6: Evaluation of Prototype II tools (from D3.8) .................................................... 22
Figure 2-1: UPDM Mediation MDWorkbench Support ..................................................... 26
Figure 2-2: MOF Profile/Metamodel to OWL Ontology "Productivity" Tool Chain .......... 26
Figure 2-3: UPDM Mediation instantiation into the Jazz/DM Tool-Net ......................... 27
Figure 2-4: UPDM Sharing – Preparation and Initial State ............................................... 28
Figure 2-5: MDWorkbench UPDM Data and Diagrams Storage ....................................... 29
Figure 2-6: MDWorkbench System Architect Tool-Net Connection ............................... 30
Figure 2-7: MDWorkbench Rhapsody Tool-Net Connection ............................................ 30
Figure 2-8: MDWorkbench Rhapsody Data and Diagram Import ..................................... 30
Figure 2-9: Tool-Net Constituents Sharing at different depth levels ............................... 30
Figure 2-10: SoS optimization workflow ....................................................................... 45
Figure 2-11: Communication system technical internal block diagram ....................... 46
Figure 2-12: Optimal communication system architecture ............................................. 47
Figure 2-13: IWTS High Level Architecture Building Blocks ........................................... 48
Figure 2-14: IWTS model ............................................................................................... 49
Figure 2-15: GUI for architectural optimization ............................................................. 50
Figure 2-16: Pareto-optimal solutions for IWTS test case ............................................. 50
Figure 2-17: Back-annotation of the minimum cost architecture ................................... 51
Figure 2-18 CAE SV1 view .......................................................................................... 54
Figure 2-19: Simple Rewriting Rule adding a District to the SoS model ....................... 55
Figure 2-20: Select an existing Contract ....................................................................... 58
Figure 2-21: Edit a Contract ........................................................................................ 59
Figure 2-22: Link Contract to model element ............................................................... 60
Figure 3-1: Components of the designed IWTS .............................................................. 63
Figure 3-2: Temperature and rainfall during simulation for Israel. The red bar denotes the point when the data is generated by the learned model .............................................................................. 63
Figure 3-3 The Simulation of the IWTS as seen in DESYREII ........................................ 64
Figure 4-1: Installing the Rhapsody plugin for semantic mediation export/import SysML models .......................................................... 67
Figure 4-2: Applying this plugin as a profile to a project ............................................... 67
Figure 4-3: Activate the export command ............................................................ 68
Figure 4-4: Setting up user credentials and target server ................................................ 68
**Figure 4-5:** Statistical Model Checking tool chain in DANSE ................................................................. 71
**Figure 4-6:** DANSE System of System Design & Simulation Workflow ......................................................... 74
**Figure 4-9:** GCSL in DANSE ................................................................................................................... 78
**Figure 4-10:** GCSL in DANSE with CAE WayForward .................................................................................. 81
**Figure 4-11:** Fault-based test generation within the DANSE tool-net. ........................................................... 84
**Figure 4-12:** Integration of realtimeAnalyzer ............................................................................................. 85

**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. 90
**Figure 5-2:** Version identification of components of the profiles including the SMC plugin profile. ............. 90
**Figure 5-3:** Activate the export command ................................................................................................. 91
**Figure 5-4:** Setting up user credentials and target server ............................................................................ 91
**Figure 5-5:** Successful export done ........................................................................................................... 91
**Figure 5-6:** Create a new UPDM project and apply to it the DANSE plugin ..................................................... 92
**Figure 5-7:** Apply the SMC import command ............................................................................................... 92
**Figure 5-8:** Enter user credentials, source server IP and URI of an element to import. ............................... 92
**Figure 5-9:** Populate a diagram from the created model for this import. ...................................................... 92
**Figure 5-10:** End result of this import. .......................................................................................................... 93
**Figure 5-11:** UPDM model in the Rhapsody Explorer that is exported ............................................................ 94
**Figure 5-12:** Same model, now imported to Rhapsody as a SysML project, after mediation. ......................... 95
**Figure 5-13:** Installing Protege 4.3 ............................................................................................................... 99
**Figure 5-14:** Control panel of the SMC plugin to Protege ............................................................................. 99
**Figure 5-15:** Protégé editing session with an ontology from SMC. ................................................................. 100
**Figure 5-16:** Example of MDWorkbench documentation ............................................................................. 101
**Figure 5-17:** A page from MDWorkbench semantic mediator tutorial ........................................................ 102
**Figure 5-18:** Example SMC client to demonstrate the SDK .................................................................... 103
**Figure 5-19:** Javadoc documenting the SMC client API ............................................................................ 104
Tables

Table 1-1: Outline of prototype cases described in this document..................................................19
Table 1-2: DANSE solutions compiled into the solutions wiki page..............................................21
Table 1-3: Usage of solution methods by DANSE industrial partners. .............................................23
Table 1-4: Tracing users requirements for the final prototype technologies.................................24
Table 2-1: DANSE profiles components.........................................................................................56
Table 4-1: Mapping from SysML to FMI.......................................................................................66
1 Overview, Purpose and Scope

The final DANSE prototype report is the summery of the tools and technologies developed during the DANSE project, used to facilitate the DANSE methodology (D4.4) [50] and to drive the DANSE solution methods, and enable the execution of the DANSE test cases (D.3.8) [51]. This is a follow up to the 3 DANSE prototype reports (D8.3, D8.4 and D8.5 [52] ) and the last revision of this sequel with an added conclusion and future plans as perceived and undertaken by the developers of the technologies and tools.

As a follow-up on the last report on the DANSE prototype (D.8.5 – on M33), some final development of some of the tools are reported here, while for some of the tools and technologies no further development took place. We also need to admit that test cases by the industrial partners took place mostly in the late periods of the project and contributed with feedback and experience that drove some of the latest improvements with the tools and technologies. Two facilities were used to help manage this interaction between users and providers; One is the exploitation forum (D8.7.3) [53] which reflected the vibrant interactions that lead the progress in these test cases. The second tool is a “issues tracking” facility (https://www.danse-ip.eu/redmine/projects/danse/issues) enabled on the DANSE wiki server and which served to report and work out bugs and problems that seemed improper to follow on the public Exploitation Forum. The issues tracking facility use within DANSE was established only during the final months of the project.

An important companion material complementing the technology information is the “Solution Methods Guide” 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.) has been updated throughout the project to the last months. Realizing 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 III and will repeat all information from the previous document that is still true today. The level of details in this document is limited by intention as the actual technologies are discussed to full depth in respective work package deliverables, but from this document the reader will have a clear appreciation of the depth and breadth of the work done on the technology side by the developing partners in this project.

The work flows among the tools in a networks of integrated tools, is a target of the consortium to achieve with this (on going) 4th iteration, and has been reflected in the new wiki “Solution Methods Guide” page(s).

The following figures render the tools and technologies connectivity status at this point in time in the end of the project. We also include a diagram of the status of this network through all of the prototype iterations.
We 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](image)

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.

Figure 1-3: The final DANSE tool-net prototype

Present connectivity diagram is in the next figure, depicting how far vendor partners have gone to reach an ideal as depicted in the figure next to it, a schema of a shared model information “bus” that connects all tools, although in our reality that is not always automated and working with the tool-net integration platform.
To facilitate rapid dissemination of the technologies among DANSE partners and users, DANSE partners use wiki pages, the DANSE SVN servers, the exploitation forum (which is public), and an issue-tracking facility on the REDMINE wiki facility — that is used internally only. 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. Complementing this is the issues tracking facility for bug resolutions that infest mostly newly developed tools and technologies — the type of tools that a research project such as DANSE produces.

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><strong>SoS Modeling Tools</strong></th>
<th>Tools used to work at the SoS level</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constituent System Modeling Tools</strong></td>
<td>Tools used at the constituents level, models of the constituents making up the SoS. 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>Joint Simulation and Analysis Tools</strong></td>
<td>Tool-net interoperability, integration and collaboration platform is</td>
</tr>
</tbody>
</table>
using semantic mediation to facilitate sharing of models among different tools in the DANSE ecosystem, and automation of processing stages in the DANSE methodology.

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

<table>
<thead>
<tr>
<th>Tool Form</th>
<th>Indicates if the tool is OTS, independent tool, or part of another tool, being an extension or new capability to that tool.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Also Requires</td>
<td>Indicates which other tools are required for using it.</td>
</tr>
<tr>
<td>Integrated by</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>User level</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>Version</td>
<td>Status</td>
<td>Date</td>
<td>Page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>Final</td>
<td>2015-02-15</td>
<td>16 of 115</td>
<td></td>
<td></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><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 and Enterprise Architect</td>
<td>Represents SoS architectures using NAF, and integrated with the Rhapsody UPDM through semantic mediation on the tool-net platform. The second tool included here is a late addition to the system in the last period.</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 and 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 profiles</td>
<td>Repository of various UPDM profiles for architecture patterns, architecture optimization, GCSL, Rhapsody extensions, simulation, etc.</td>
<td>Rhapsody Profiles</td>
<td>Rhapsody</td>
<td>Rhapsody</td>
<td>Engineer</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><strong>Constituent Systems Modeling Tools</strong></td>
<td></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>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>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>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>Dymola</td>
<td>Perform statistical model checking on the results of a simulation</td>
<td>Standalone</td>
<td>DESYRE</td>
<td>DESYRE</td>
<td>Engineer,</td>
</tr>
<tr>
<td>Simulink FMU exporter</td>
<td>DESYRE joint simulation</td>
<td>Standalone</td>
<td>None</td>
<td>Tool-Net</td>
<td>Power user</td>
</tr>
<tr>
<td>PLASMA statistical model checking</td>
<td>DESYRE joint simulation</td>
<td>Standalone</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>TestCast test generator</td>
<td>TestCast test generator</td>
<td>Standalone</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>Contract-based run-time verification</td>
<td>Contract-based run-time verification</td>
<td>DESYRE Analysis</td>
<td>None</td>
<td>Tool-Net</td>
<td>Engineer</td>
</tr>
<tr>
<td>GCSL Contracts Analysis Tool</td>
<td>GCSL Contracts Analysis Tool</td>
<td>DESYRE Analysis</td>
<td>DESYRE</td>
<td>Tool-Net</td>
<td>Engineer, Power User</td>
</tr>
<tr>
<td>Fault-based Test Generation Tool</td>
<td>Fault-based Test Generation Tool</td>
<td>DESYRE Analysis</td>
<td>DESYRE</td>
<td>Tool-Net</td>
<td>Engineer, Power User</td>
</tr>
<tr>
<td>Timing analysis</td>
<td>Timing analysis</td>
<td>Stand alone</td>
<td>Rhapsody, MARTE</td>
<td>None</td>
<td>Power user</td>
</tr>
</tbody>
</table>

1 DESYRE is serving as a hub of many technologies in this table, and integrate also the PLASMA tool.
### Table 1-1: Outline of prototype cases described in this document

<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 Client</td>
<td>Client tool to generate multiple instances of model elements using “Concise” methods.</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.
Figure 1-5: One of the work-flows cases discussed in the solutions wiki page.

The following **Table 1-2** 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 environment</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. D3.8 [51] reflects on the 3rd prototype. We draw in the following Figure 1-4, the tools evaluation as per users’ evaluation following prototype iteration II.

![Figure 1-4](image)

**Figure 1-4:** Screenshot of the tools evaluation as per users’ evaluation following prototype iteration II.

The assessment is according to the following convention:

- **0** poor
- **1** average
- **2** good
- **3** excellent

A (not so final) table taken from the forming up D3.8 is on the 4 user partners evaluation of the different solution methods 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>1.0</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>1</td>
<td>1</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>1</td>
</tr>
<tr>
<td>11</td>
<td>Joint simulation</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>Statistical model checking</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>Formal verification</td>
<td>1</td>
<td>1</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 1-6](image)

**Figure 1-6:** Evaluation of Prototype II tools (from D3.8)

A (not so final) table taken from the forming up D3.8 is on the 4 user partners evaluation of the different solution methods in the next table.

<table>
<thead>
<tr>
<th>Nbr</th>
<th>Solution Method</th>
<th>IAI</th>
<th>CARMEQ</th>
<th>THALES</th>
<th>Airbus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Model SoS</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Abstract CS model</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>Apply architecture patterns</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Generate architecture alternatives</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Activity</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Generate optimized architectures</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Perform joint simulation</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>Perform statistical model checking</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td>Evaluate emergent behaviour</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>Evaluate goals and contracts</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>Perform formal verification</td>
<td>x</td>
<td></td>
<td>(x)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Configure DANSE Tool-Net environment (*)</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Share models</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 1-3: Usage of solution methods by DANSE industrial partners.

A traceability of these with technologies introduced into the DANSE prototypes are presented in the next table.
The rest of this document is an account to the different tools and technologies in the DANSE prototype - one chapter (or rather a sub-chapter under the 3 main categories of tools in DANSE) per tool or technology. Since most of the tools and technologies are unique in each one’s function, usage and value, so are different conclusions and lessons learnt about each of them provided at the end of each chapter for each of the tools.

In addition a final conclusions section at the end of the document is added to kind of summarize over all items in this document in a general way.
2 SoS Modeling Tools

2.1 Rhapsody

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 Rhapsody that are required in wider contexts as described per other technologies in the prototype.

As an SoS design tool, Rhapsody'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 Rhapsody 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 Rhapsody 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 Recent Improvements and new features

1) UPDM2:
   2.0.2 contains all the UPDM2 improvements developed in year2

2) Genericity:
   The previous version of Rhapsody Tool-Net Client was deeply focused on UPDM.
   For importing constituent from the Tool-Net it was requiring an UPDM Port on the server in order
   to list the available views to user.
   In year3 Sodius made great improvement on the genericity of its client and on the semantic
   mediators. Most of UPDM dependencies are removed from Rhapsody Tool-Net Client and the
   list of available views on the server is provided by a Rhapsody GET Port. The SODIUS
   Semantic Mediator for Rhapsody is now able to provide a quick preview of the mediation, in
   addition of a full mediation. This allows Rhapsody Tool-Net Client to only deal with Rhapsody
   models, extending its usage to future semantic mediation based on other ontologies than
   UPDM2.
   A tutorial exists showing how to use Rhapsody ToolNet Client in another context than UPDM
   mediation.

3) Performance improvement and fixes
Several additional resources were creating for helping advanced user for developing their own solution. A tutorial with a sample describes a full bidirectional semantic mediation solution from Rhapsody and a new ontology dedicated to action flows, from the definition of the ontology itself, to the writing of rules mediations.

2.1.5 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.

![Figure 2-4: UPDM Sharing – Preparation and Initial State](image)
• **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.
According the elements chosen at the System Architect sharing step, information can be shared at different levels between partners.

Figure 2-9: Tool-Net Constituents Sharing at different depth levels

2.1.6 Conclusions

The Tool-net clients developed for IBM Rhapsody, IBM System Architect and Sparx Enterprise Architect offer to customers a generic and interactive solution for connecting tools on the Tool-net. Based on a common platform, it shows that a large variety of products can be integrated in the Tool-net, with a limited effort.
Rhapsody client is not restricted to UPDM ontology and can be reused to other domains of mediation. It aims to be user-friendly, browser oriented, and allows users to interact with the mediation by a simple selection of elements to handle.

The semantic mediation behind each of those clients demonstrate the capability of model sharing between heterogeneous tools. Through the Tool-net, different partners with different tools in different locations can contribute to the definition of an SoS, with UPDM. Several patterns in the Tool-net were tested (direct access to repository, Post or Get Port mediation, duplicate port names...) for successfully demonstrating the maturity of the platform.

**2.1.7 Lessons Learnt**

We have developed tool clients which are probably too much linked to UPDM mediation. A tool client for the Tool-Net needs to be as generic as possible in order to avoid an explosion of dedicated clients for each purposes. Also most of the intelligence should be on the server, on the semantic mediation side. Knowing which kind of elements to display from a user model, which one to share, and how to visit them - should be known by the server itself. A such behavior would improve both performances, reusability of the tool Client, and maintenance (client side would not need any update).

**2.1.8 Future Plans**

Semantic mediation capability should be extended to other kind of UPDM views and become bidirectional for System Architect and Enterprise Architect. Versioning, updates and notifications are clearly a big priority for having a perfect industrial solution.

**2.2 System Architect and Enterprise Architect**

These are off the shelf tools 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].

![Figure 2-12: SoS Architecture Patterns in the DANSE Methodology Overview](image-url)

<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>2015-02-15</td>
<td>34</td>
</tr>
</tbody>
</table>
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).
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.

Figure 2-13: Six-stage process showing architecture pattern process
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
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 Conclusions

Applying architecture patterns can only be as good as the richness of the patterns repository allows. For best use, the repository should be enhanced on a regular basis to include any new patterns that are discovered. Pattern mining is a “soft systems” art. The elements that make an architecture pattern useful are the architectural elements found in the pattern template. Therefore, discovering new patterns involves interaction with the SoS users to determine the capabilities, activities, technical quality and emergent behaviour, as well as interaction with the SoS and CS architects to determine the architectural structure and the technical details of activities and functions. These interactions can often be performed through focus groups, with a qualified facilitator seeking to elicit the desired information from the subject matter experts.

Architecture patterns contain two imperative fields, architecture structure and performance metrics, which are used in an optimisation environment such as Concise Modelling. The architecture structure provides the connectivity of the system exposing the relationships between key parts and offering the basis of the model; the performance metrics describe operational/behavioural constraints and capabilities. Architecture patterns can be seen as a supplementary input into the optimization phase of the DANSE methodology. It has been shown in the preceding section of this report how architecture patterns can be fed into the Concise Modelling process to provide the initial model for architecture optimization.
2.3.6 Lessons Learnt

Applying architecture patterns is an extremely important process and relies on an ability to recognise standard architecture patterns as well as being able to recognise what kind of pattern is needed to meet a set of requirements. An experienced systems architect will have some idea of the type of pattern necessary for application, but they may not have considered every possible option. As the library continues to grow and a larger number of patterns are added, the variety and quantity of patterns will increase – resulting in an even more useful resource for both systems architects and systems engineers. The following are regarded as good practice and tips for architecture pattern application:

1. Identify what are the local and global goals of both the SoS and its constituent systems. These goals can be expressed in the form of requirements (functional and non-functional requirements) and constraints.

2. Identify what architectural elements are in scope (with reasons/justification) of the design of the given scenario – e.g. command structures, communication infrastructures, security and resilience features etc.

3. Identifying key constituent systems or parts of the system can be helpful for when searching the repository by keywords or phrases. For example, an architect designing a water distribution system may be thinking of including water towers, or dams. Searching these keywords will narrow down the number of patterns to those which include these elements.

4. Identify the key features/characteristics of the interfaces of the constituent systems.

5. Be aware there may be more several architectural alternatives which may suit a set of requirements and be fitting for the SoS design – the first architectural solution may not always be the best.

6. Keep an open mind – SoS architecting is a skilled and creative process that frequently draws upon years of experience, the repository aims to capture such knowledge and experience and make it readily available as possible.

7. Downloading and applying a pattern to a project is straightforward (see Deliverable 5.2 for details). However, the architect must recognise the pattern applied may be a high-level representation of the pattern and may need manual modification in Rhapsody once applied.

8. When modifying a pattern and creating a slight variation of the pattern, it is good practice to store this new pattern back into the online repository as a new record. The ‘Author’ field should be selected and the author identified.
9. Typically, when creating an SoS architecture from scratch, the UPDM modelling methodology used in DANSE is a good reference. Starting with the operational views, moving on to the functional and systems views, applying patterns where suitable as the architecture evolves.

10. The repository is an ever growing resource and does not contain patterns to cover every scenario, however root architecture patterns are commonly found in all SoSs, therefore high level patterns for communication, command and control will always be relevant. As more patterns are added from other domains, the richness of the repository will provide a great resource for systems engineers and architects.

### 2.3.7 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 automatic pattern searching techniques.

To complement the online systems architecture patterns library, and to bridge the gap between an unmanageable number of patterns and developers searching for a pattern for their current design problem, system architecture patterns ontology is currently being considered. An architecture patterns ontology will further improve the usability of system architecture patterns and help engineers, especially inexperienced ones to adopt architecture patterns in their designs.

### 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

2.4.3.1 CAE

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.
2.4.3.2 IWTS

The Integrated Water Treatment and Supply System (IWTS) is the name for the array of resources, assets and facilities used for supplying water from different resources to different types of customers. Its purpose is to provide quality water to its consumers in an undisrupted manner. The test case description considered following scenarios of IWTS operation.

1. IWTS Operational Reference,
2. Failure Scenario,
3. Drought Scenario,

Fehler! Verweisquelle konnte nicht gefunden werden. summarizes DANSE optimization techniques used in this test case.

The IWTS SoS consists of the following subsystems:

- Control centre
- Pipe networks
- Water sources
- Reservoirs

Each water source is considered to be one of following types:

- Desalination facility
- Ground water source
- Grand National Lake
Desalination facilities and ground water sources are also classified by their size. Desalination facilities could be either medium or large and ground water sources could small or medium.

Each type of water sources is characterized by following data:

- Maximum water capacity
- Fixed operational cost
- Supply dependent operational cost
- Founding cost

Data also exists for the current water supply of existing water sources and possible places and types for new water sources.

Global SoS optimization goals are:

- Minimize operational cost,
- Maximize water supply capacity.

Each supplier has the local optimization goal of maximum profit.

The test case depicts the desired long term evolution scenario of the water supply capacity. The current water supply system is considered as a starting point. In the test case, the water demand grows from the
current level to 10% more. The future IWTS architecture must satisfy the rising demand while keeping the operational cost at the current level.

The founding cost is constrained by the available budget. The goals are achieved by applying architectural decision of following types:

- Decision on adjustment of water supply for existing water sources,
- Decision on founding new water sources,
- Deciding the water supply capacity for the new water sources.

The Concise Model for this example is shown on Figure.

![Figure 2-14: IWTS model](image)

This test case represents the following SoS optimization techniques and extensions:

- Pareto Multi-Objective optimization is used to minimize the total system cost as well as maximize the water supply capacity.
- Black-box & simulation integration extension is represented by IWTS parameters calculated based on the simulation of water sources.
- Partial freezing extension is represented by providing information on existing IWTS and is used to fulfil Dynamicity and High scale requirements in SoS optimization.
- Concise modelling technique is used to represent all water sources.
- Patterns are used to define centralized or decentralized management systems.
• Semantic mediation extension used to convert UPDM concise model into SysML and to convert SysML back-annotated optimal models back into UPDM.

We developed a new user interface to run the architectural optimization solver, shown in Figure 2-15.

![Figure 2-15: GUI for architectural optimization](image)

Five Pareto-optimal optimal solutions are found for the test case data are shown on Figure 2-16. Two potential solutions have been selected by the user. One of the solutions minimizes the operational cost and another maximizes the water supply. In order to minimize the supply cost, future water supply capacity is kept on the lowest allowed level. In this case the supply cost is lower than the current supply cost by 9% with almost 40% increase in water supply versus the current water supply. The second solution demonstrates that the water supply could be increased further by approximately 7% within the available budget while keeping the supply cost on the current level. The back-annotation of the second solution is shown on Figure 2-17.

![Figure 2-16: Pareto-optimal solutions for IWTS test case](image)
2.4.4 Conclusions

AOW produces high quality SoS architectures and could be seamlessly integrated in DANSE SoS development methodology. AOW ensure “correct-by-construction” architectures that eliminate expensive development cycles caused by late detection of problems in architecture.

2.4.5 Lessons Learnt

We recognized several important features looking by users. The first one is user’s confidence in the optimal solution in uncertain environments especially during long lifecycles of SoS-s. To immunize solution given by AOW against uncertainty of system parameters we developed Robust Optimization Methodology for AOW. However, this methodology is hard for implementing by users without optimization background. Hence, we need to develop representation of uncertain parameters and automatic generation of Robust Counterpart within existing concise modelling framework. The second one is further reducing modelling overhead required from user. In real-life necessary models are often not fully known. The System Engineers deal with raw data containing big number of parameters. We recognize necessity of automatic or semi-automatic conversion of such data to a model. Another necessary feature is optimization of dynamic behavior of the target system, and tacking into account system behaviour on the step of system architecture design. We also recognize necessity of simplification of user constraint definition and developing automatic techniques for linearization of non-linear user constraints (if required by the optimization solver). Furthermore, metric library provided with current AOW tool should be extended to provide various reusable metrics for particular domain-specific constraints. We also need to standardize metric developing process to allow domain engineers write their reusable domain specific metrics without optimization expert.
2.4.6 Future Plans

AOW is available for customers on service contracts. We plan to extend current AOW according to the lesson learnt, i.e., allow uncertain representation of parameters, automatic model retrieving based on customer’s data, and, in longer future, move AOW maturity level to a standalone product for Systems Engineers without optimization background.

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 already presented in the first prototype [42] and will be detailed here.

2.5.1 Purpose

The graph rewriting rules (or Graph Grammar) are used to model the dynamic reconfiguration of the SoS structure and the participating CSs. With this formalism the designer is enabled to model evolutionary aspects of the SoS. As a simple example consider the changing infrastructure of a city within several years. Technically, the investigation of subsequent SoS architectures is based on UPDM and IBM Rational Rhapsody. The designer is enabled to specify evolution within Rhapsody using the DANSE profile enabling the generation of a new architectures out of an initially existing one. This approach represents the implementation of the solution method “Generate Architecture Alternatives”.

2.5.2 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 techniques are needed to model the SoS model, the third to model the evolution of an SoS, while the GROOVE generator is used as a backend to generate architecture alternatives.

2.5.3 Demonstration Scenario

The following example is an excerpt from the DANSE D3.3.2 Concept Alignment Example Description deliverable and slightly adapted for the purpose of this document.

The CAE behavioural part represents 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 2-18.
Figure 2-18 CAE SV1 view
The evolutionary changes are modelled by using rewriting rules like illustrated in Figure 2-19 where a new district is added and connected to the CAE.

![Figure 2-19: Simple Rewriting Rule adding a District to the SoS model](image.png)

A set of rules defines the evolution model of the SoS. 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-backed to the Rhapsody tool.

### 2.5.4 Conclusion

With the extension of the DANSE profile with respect to rewriting rules and the automatic transformation of UPDM Rhapsody models to the backend tool Groove we have realized a powerful mechanism to model changing aspects of SoSs, which was not possible so far. This technique enables the designer to model evolution which is a crucial aspect of SoSs. With this, future architectures can be explored which would harm constraints on CSs and the SoS. As an example, consider that a fire department cannot reach any road segment after some infrastructure changes occurred.

### 2.5.5 Lessons Learned

So far, the modelled rules can be annotated with some weights, such that some rules are triggered more likely than other rules. Further, the exploration of the architecture alternatives was bounded by a certain number of reached architectures. Experiences have shown, that this simple approach can help to determine critical architectures, which violate crucial CS or SoS constraints.

### 2.5.6 Future Work

Future works include the extension of the rewriting rules such that the designer is enabled to affect the architecture generation process in a more interactive manner than in a complete automatic fashion as it is realized so far. This semi-automatic process would help to find critical architectures more efficiently.

### 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.

2.6.1 Conclusions

The organization of all DANSE profiles and plugins into a single structure has shown very useful for the project and ensuring a common base to be shared among all users. It constitutes a kind of a DANSE version of its prototype for the technologies that are embedded in and through the Rhapsody extensible tool.

2.6.2 Lessons Learnt

The maintenance overhead is small and the value it significant.

2.6.3 Future Plans

None, as the contents of this collection is actually the responsibility of individual contributors to DANSE and after the conclusion of the project and dissolution of the bidding contracts, this collection cannot be managed anymore by any responsible and authorized body.

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 fields are check while editing for syntax errors and references to elements of the model.

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. 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 2-20).
In Figure 2-21 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 has to select an Anchored Element in the Feature Dialog within Rhapsody (see Figure 2-22). The anchored element is the component which shall satisfy the Contract.
2.7.4 Conclusion

The implemented tool to specify GCSL constraints was summarized here in short. It is able to support the designer to specify correct constraints by offering a GCSL pattern list and performing a syntax check on the specified constraints.

2.7.5 Lessons Learned

Experiences show that the usage of the editor helps in the specification of constraints. It offers the user to select constraints from a list of GCSL pattern and automatically performs a syntactic validation.

2.7.6 Future Work

The GSCL editor will be used in future projects. In the context of future projects it will be extended by further constraint patterns.
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.

Rhapsody’s connectivity with the Tool-net enables, like what happens with the DESYRE tool – to enable automation flow of artefacts from tools that are not integrated. For instance, the FMU objects that are needed by DESYRE to do joint simulation, can also originate in Rhapsody SysML models, much as they can come from other simulation capable tools (Simulink, Modelica, etc.). The GCSL editor is integrated via this Rhapsody tool as tagged stereotypes of GCSL statements. That is also mediated form this tool on to the Tool-net integration platform, on with semantic mediation, and reaching through an automated flow to the DESYRE for contract analysis and validation.

3.1.1 Conclusions

This is a very useful and friendly tool much that engineers working in MBSE like to use, its extensibility has proven very useful in DANSE, and its integration into the Tool-net enables the DANSE tools integration vision to meet reality and cover technology gaps that some of the tools presented.

3.1.2 Lessons Learnt

This is an excellent tool to work on new technologies and capabilities for quick and easy access to the systems engineer working with MBSE.

3.1.3 Future plans

DANSE would have continued to adopt this tool for its stated goals.

3.2 Abstraction tools – Statistical Learning

3.2.1 Purpose

Statistical Learning is an abstraction technique that allows to learn the behaviour of a set of input streams by deriving stochastic quantities from them. The method works best for stochastic input streams which exhibit the Markov property, but will produce also good approximations if full Markov property is not guaranteed.

The method is best used for the abstraction of very complex systems, especially since in these cases a statistical abstraction is preferred compared to a big and slow detailed model. Good examples are climate...
conditions like wind, temperature or rainfall. Another typical application scenario is the abstraction of legacy systems, which can be replaced by statistically learned models.

The method just requires an input stream to learn from, while not exceeding a certain area of the state space. The state space has to be limited beforehand, but aside from that no further modifications to the method are necessary. This results in a good reusability of the method, for example by learning and comparing the systems results under the influence of different environmental conditions. Most of the time this is achieved by simply exchanging the data-files used as input. In these cases the model itself does not need to be touched at all allowing for a quick simulation under new conditions.

Since the statistical learning is used during simulation runs it is able to adapt to new incoming data. The method first learns from the input signals during a learning phase. During this phase the original data is given as an output, but as soon as the learning reaches certain accuracy (or simply the end of the data-file) the learned model is used to replace the input data and provide statistically identical data for the simulation.

### 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 like state space imitations or the point in time when the learned model should replace the input. 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. After that the “Statistical learning”-FMU could be used within DESYREII (see chapter 4.4) for simulation.

### 3.2.3 Demonstration Scenario

For demonstration a simple IWTS was designed. This IWTS reacts on environmental properties like rainfall and temperature. Such environmental conditions need to be included into a system to evaluate the natural resources for the IWTS as well as the demand for water by the customers.
The statistical abstraction provides a very easy way to create these conditions as inputs to the IWTS. In this example two environmental conditions are used: Rainfall to refill lake- and groundwater and temperature to determine the customer demand. This will guaranty high refill rates during wet-seasons and high water demands during summer.

Rainfall and temperature are learned as soon as the simulation starts. At some specified point of the simulation the data-files are replaced by the learned model. This allows for an endless simulation without repeating data.

The simulation could be done in Simulink or in DESYREII by also creating the model structure in Rhapsody. All necessary parameters for the statistical learning FMU could be adjusted in Rhapsody.
3.2.4 Conclusions

In DANSE the method of statistical learning is realized as a FMU that could be integrated into models to learn statistical behavior from data or CS and creating an abstraction. The FMU was generated by using Simulink and the DYMOLA FMU export. This allows for a simulation in DESYREII using the DANSE Toolchain. As an example the environmental conditions for an IWTS were learned.

3.2.5 Lessons learnt

Before DANSE the method of statistical learning was already used in physics in many different ways: As a tool for system analysis, system description and rarely as a simple way to derive a model that allows for a Monte-carlo simulation. In DANSE the method was refined by focusing on the model generation capability of the method. Therefor it was introduced into the DANSE Tool-net to allow its integration into FMU based systems, but also the method itself had to be adapted in a way that allows for an easy introduction into an already given system. In physics the method is mostly used on always completely available datasets, but in DANSE the method is used on incoming data streams allowing only access to their current or most recent values. This problem was solved by continuously adapting the learned model to the incoming data. Since the learning is usually done during a simulation run it had to be designed in a way that would allow the simulation to run properly without any disturbance even during the learning phase. As soon as the learned abstraction reaches a certain accuracy, a defined time stamp or the end of a data file used as input, the model is activated and therefor replacing the original source.
3.2.6 Future plans:

The statistical learning is now available for use in many different application scenarios, but could also be refined further. One problem currently occurring is that the method requires exponentially more memory with an increasing number of input streams. In the FMU created in DANSE the learning is therefore limited to three correlated input signals. This limitation could be lifted by using proper n-dimensional interpolation and fitting algorithms. This method is very effective, since the underlying model components are not independent from each other and often follow simple polynomial behavior.

OFFIS is very interested in using the method in the context of automotive transportation as a means to learn the behavior of drivers and autonomous systems alike. In this context the learned abstractions will be used for analyzing driver reactions on different levels of automation as well as a means to provide abstractions for simulation purposes.

3.3 Modelica w/ System Modeler

Off the shelf tool by Wolfram® - 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)

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) .

3.3.1 Conclusions

This tool was not used since none of the industrial users had any assets in Modelica to use. So no conclusions about the tool or the Modelica language can be drawn at this point.

3.3.2 Lessons Learnt

None

3.3.3 Future Plans

N/a.
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;FMUParameter&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 Rhapsody to export (SysML or UPDM) projects in RDF to the tool-net semantic mediation container (SMC) 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 capability.

4.1.2.1 Demonstration scenario

1. A Rhapsody project is equipped with the smc 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 smc 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.

![Figure 4-3: Activate the export command](image)

![Figure 4-4: Setting up user credentials and target server.](image)

### 4.1.3 Conclusions

The FMU export facility is an excellent capability that has been found to be very useful. This is still evolving standard and work on extending it to discrete events rather than continuous signals with jumps has been performed during the project to fit with what has been an identified need by DANSE users. FMU standardize the representation of the behavior of components so that different tool can produce that form their own internal formulation, while consumers of the FMU objects so not care where the object came from.

The integration into the tool-net as a linked resource on the internet makes it simple to integrate and automate work flows that feed from the semantic mediation container (SMC) platform where all necessary data for a joint simulation can be obtained. The SMC export facility simply takes one step forward in also tying up the FMU object with the standard RDF formatted model that Rhapsody exported, keeping the correct relations of the model and the FMU object. That makes it less error prone compared with the situation...
where the FMU (see for instance the case of Simulink via Dymola below) needs to be linked to the model using manual web tools.

### 4.1.4 Lessons Learnt

The approach presented here for the Rhapsody seems to be the right approach, and it needs to be applied to more tools. In the DANSE case, there are several tools that can produce FMUs and link them automatically, but more tools need to do so and enable an automated flow. The implementation of this facility to the DESYRE tool enabled this tool as a consumer of models and FMU blocks to perform a fully automated access to all needed data to do joint simulation.

In addition, the capability for automatic import of external FMUs into Rhapsody has been found extremely useful, as it significantly reduce modeling effort and minimize chances for errors. In this way a complex system model can be easily build from black-box FMU models exported for other domain specific tools.

### 4.1.5 Future Plans

The FMU link to RDF models will be pursue via standardization bodies such as OMG working group OSLC 4MBSE which works on extending the use of OSLC in MBSE, and in the Modelica organization.

Future plans per FMU include also extending its coverage when moving on to FMI 2.0, enhancing the continuous event handling that has started already.

Most of the current work was done using so called FMI model-exchange mode, we plan to extend it to FMI co-simulation mode. Co-simulation mode is very powerful approach, that impose less semantics constrains, so potential could be used for very wide range of tools. However some more features need to be added to FMI co-simulation, specifically to handle models with discrete event-based semantics.

### 4.2 Dymola Simulink FMU export

#### 4.2.1 Purpose

Generate FMI executable models (http://www.fmi-standard.org) from Simulink and Modelica models.

#### 4.2.2 Tools and environment

Dymola is an off-the-shelf tool based on the Modelica standard language (http://www.modelica.org). Dymola has an FMU export feature to generate FMI executable models from of 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/
4.2.3 Conclusions and Lessons Learnt

DYMOLA allows for an export of Simulink models as FMUs, but it has certain limitations. One limitation is the use of only real variables. Integrating the generated FMU into already existing systems could require additional FMUs solely converting in-/outputs into different variable types, and of course these cannot be generated using DYMOLA. Another problem is that DYMOLA automatically selects almost all variables for the FMU’s model description-file. As an example: The FMU containing the abstraction method "Statistical Learning" is heavily using matrix calculations internally. Each matrix-element is taken as a variable for the model description-file and this is done for each operational block in the Simulink model resulting in several hundred thousand entries into the model description-file. This is not only slowing down the exporting process, but you also need to erase the huge not-necessary content in the model description. Another issue is that not all of the MATLAB-functions can be used in the corresponding Simulink-block. If there is a not supported function, it will be ignored without a warning.

4.2.4 Future Plans

N/A.

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 too big to be analysed with model-checking, whereas since SMC just simulates the model a large number of times and checks the contracts for each simulation. The precision of the estimation increase the number of simulations. Mathematical results in statistics allow us to compute the number of simulations to perform in order to reach the precision requested by the user.

4.3.2 Tools and environment

PLASMA-LAB is distributed as SMC library or a standalone tool. In the DANSE 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.

![Figure 4-5: Statistical Model Checking tool chain in DANSE](image)

As already explained in Subsection 4.4.3, the DESYRE collects the compiled model from Rhapsody. Similarly, it also gets the contracts attached to this model. These contracts are 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 hand written specification but with a formal semantics whereas LTL is a very low level language and LTL is generally understood by experts only. Whenever the user starts the SMC analysis [24] goal is translated by an external compiler invoked by DESYRE to produce an equivalent LTL formula, which is then passed to PLASMA-LAB. Then, PLASMA-LAB starts and asks DESYRE to start the first simulation. PLASMA-LAB asks for new steps of the simulation until the contract to check is evaluated to either true or false, at which point it stops the current simulation. This step is repeated as long as PLASMA-LAB requires more simulations to attain the requested precision. When the session finishes, PLASMA-LAB returns the results collected by DESYRE.

### 4.3.3 Demonstration scenario

The PLASMA-LAB/DESYRE combination can be demonstrated over two test cases of the project, namely the IWTS from IAI and the ATM from Thales.

For the IWTS test case, one of the goal is to ensure that each customer gets enough water. This can be formalized as a GCSL contract and then to a B-LTL formula suitable for PLASMA-LAB. This property can then be checked on a large number of simulations, each of them representing a few hours of execution of the system. The results provided by PLASMA-LAB helps assessing whether the chosen design satisfy the goal.
For the ATM case study, PLASMA-LAB can be used to check a non-functional property. Here the goal is to ensure that during 25 years of executions, the system remains up at least 90% of the time. Such a property cannot be checked with classical model checking, as it requires to measure the time during which a predicate holds. Here again, the goal can be formalized as a GCSL contract, is transformed to a B-LTL formula and can be checked by PLASMA-LAB. The results helps assessing whether the system is reliable enough and can help, for instance, to adjust the amount of redundancy in the system.

### 4.3.4 Conclusions

The analysis provided by PLASMA-LAB allows checking whether a GCSL contract is met by a SoS that can be simulated with DESYRE. The result is a probability which allows the user to quantitatively evaluate whether the contracts holds. Depending on the nature of the analysis, the user sets the precision which in turn affects the analysis time. Thus the user controls the trade-off between precision and speed of the analysis. The approach is demonstrated on two industry-scale test cases, which proves its applicability.

### 4.3.5 Lessons Learnt

The DANSE project allowed us to integrate and evaluate PLASMA-LAB in an industry-like environment and scale.

First, PLASMA-LAB is able to check contracts over large scale models, and seems to scale without any problems. The main bottleneck encountered during the SMC analysis is still the simulation time.

Second, PLASMA-LAB can be integrated with an external simulator, DESYRE in our case, quite easily in principle. However, even though we designed early in the project a metamodel for the communication between PLASMA-LAB and DESYRE, we still had issues to put the SMC analysis in place. Therefore, in a similar project, we would try very early to have a working prototype.

### 4.3.6 Future Plans

We plan to improve the general performance of PLASMA-LAB by optimizing the procedure to check a contract, and decide as soon as possible whether a given contracts holds, in order to reduce the number of steps of simulation needed to decide whether a contract holds.

Another direction is to improve the feedback in case of an error, so that the final user does not need to read the logs for solving the problem. This problem has been pointed out to us by some of the industrial partners.

### 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 Purpose

In the DANSE tool-chain, DESYRE [23] provides the framework for joint simulation (i.e. simulation of heterogeneous models) based on the FMI 1.0 standard, performance evaluation and property-satisfaction analysis of Systems of Systems (SoS). DESYRE joint simulation is also integrated with the PLASMA-LAB [25] verification engine to perform statistical model checking.

4.4.3 Tools and environment

Inputs to the DESYRE joint simulation tool are: (1) the structural model representing the SoS architecture, describing how the Constituent Systems (CS) are interconnected, and (2) the behavioral models describing each CS. Behavioral models 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 the FMI 1.0 standard [32]. The FMI standard prescribes the exchange of behavioral models in the form of FMUs. The SoS architecture is provided as a UPDM [8] diagram using a set of DANSE extensions to the UPDM profile. This second part of the design flow is supported, in DANSE, by the IBM Rhapsody tool.

Both the SoS architecture and the constituent system behavioral models (exported as FMUs) can be published on the DANSE IBM Semantic Mediation Container platform (SMC), which makes them available for use by the other tools that are connected to the SMC. SoS architecture and FMUs can also be exchanged by file sharing. DESYRE is able to import UPDM designs, possibly containing multiple SoS architectures, and the FMUs for each Constituent System from the DANSE Semantic Mediation Container platform as well as from file.

The support for several architectural alternatives within the same UPDM SoS design is a very important feature in DANSE. Firstly, it allows to save the designer time by reusing parts of the UPDM design (e.g. subsets of the constituent systems). Secondly, it supports the interaction with automated techniques for the generation of architectures: such as the architecture optimization tool (IBM Concise) and Graph Grammars for architecture evolution (GROOVE [22]), thus supporting the entire SoS design and life-cycle. Finally, it allows the comparison of the architecture alternatives, by joint simulation or other available analyses, and the choice of the most satisfactory SoS architecture in terms of performance, goals achievement or other metrics of interest. For this purpose, DESYRE allows to set up a unique joint simulation configuration and apply this configuration to several architecture variants to allow an easier comparison of the results. Besides this, simulations can be saved into trace files for sharing, later inspection or further comparison.
Once the simulator and the analysis tools have been configured, the simulation can be run.

### 4.4.4 Demonstration scenario

DESYRE Joint Simulation has been applied successfully to all of the industrial test cases (AGT from Carmeq, IWTS from IAI and ATM from Thales) as well as to the CAE [38] example (Airbus).

The IWTS industrial test case demonstrated the use of DESYRE with FMUs generated from distinct tools: the Rhapsody FMU exporter, Open Modelica FMU Exporter and Matlab-Simulink FMU Exporter. It also demonstrated the integration of the different FMUs into a single Rhapsody architecture which was imported into Desyre together with the required FMUs. Joint Simulation of the IWTS SoS was used to analyse the behaviour of the desalination facility under different C4I control behaviours and to show how overall performance was affected. This was easily supported, within the FMI standard, by simply replacing the relevant FMUs. Further use of Joint simulation allowed the SoS designer to fine-tune the model parameters, in particular to identify proper water flow capacities and stochastic characteristics of the desalination facility. Matlab-Simulink FMU Exporter was found to be not fully FMI-compliant and a fixed version was provided by Mathworks.

The ATM industrial test case demonstrated the use of DESYRE to support the simulation of partial system views, by using the DANSE FMUIgnore stereotype. This allowed to manage the complexity of the test case, that required the integration of 18 FMUs generated by the Rhapsody FMU Exporter. The ATM test case modelled fault and repair behaviours of the various Constituent Systems and DESYRE joint simulation was used to provide a quantitative evaluation of the availability of the overall SoS. Then, the Plasma Statistical Model Checking tool provided a likelihood estimate of having more than one system in failure state at the same time, by aggregating statistical data collected in DESYRE simulations. While using the IBM Rhapsody FMU exporter, two bugs that made it not fully FMI-compliant were identified and a fixed version was provided by IBM.
The AGT industrial test case, on top of showing the integration of FMUs produced by different tools (Rhapsody FMU exporter and Matlab-Simulink FMU Exporter), demonstrated the full integration of DESYRE Joint Simulation with the IBM Semantic Mediation Container platform. In particular, the Rhapsody architecture was exported to the SMC in UPDM format, including URIs for FMU resources (again, available through the SMC). This UPDM architecture was demonstrated to be mediated (i.e. translated into a different format for interoperability) into the DANSE BSO architectural language. Finally, DESYRE was used to import the BSO architecture as well as to recover the FMUs from the corresponding URIs and set-up automatically the simulation environment, thus proving the greater advantage in terms of user experience and tools integration bought by the Semantic Mediation Container.

The CAE model was used to demonstrate the application of DESYRE Joint Simulation to the identification of Emergent Behaviours in an Emergency Response scenario within a large city. There, a wrong design of the communication protocol was identified to cause the miss of fire notifications and the spread of the fire in the entire city. This happened under very rare conditions of fire notifications happening in a short time frame.

### 4.4.5 Conclusions

DESYRE Joint Simulation was used extensively in the four test cases of DANSE. This allowed the project participants to stress-test the DANSE methodology as well as the various tools involved. The use of joint simulation allowed to identify bugs in existing commercial tools as well as in the DANSE tool-chain. Most of these bugs were solved and thus provided a strengthening of the FMI-standard reception as well as an improvement of the maturity of the tools involved in the project. FMI-based joint simulation of SoS models proved to be challenging in terms of successful integration of the tools as well as in terms of the complexity and efficiency of the simulation task. Some models were not developed to take full advantage of the event-based simulation support of DESYRE, thus leading to sub-optimal performances. A further challenge came out of the discovery that some interaction mechanisms between CSs could be modelled in more naturally not in terms of flow of data but in terms of services offered and consumed. This was readily supported by IBM Rhapsody flowcharts and UPDM standard ports. However it was neither directly supported by the FMI standard nor by the IBM FMU exporter plugin. Fruitful interaction between ALES and IBM led to the identification of a solution that was built on top of the FMI standard. In particular, events generated by flowcharts are serialized as strings relying on the native FMI support for strings. The IBM FMU exporter was extended to support this and DESYRE was able to import and use the resulting FMUs. This allowed the simulation of models where CSs interaction was designed in terms of service offer and consumption.

### 4.4.6 Lessons Learnt

The FMI standard can be used for effective integration of Constituent Systems FMI-compliant modelling tools such as Rhapsody, Matlab and OpenModelica. The DANSE extension of the UPDM profile and its support within Rhapsody can be used effectively to integrate FMUs from different tools into a SoS-level architecture. The DANSE extension of the UPDM is able to support different architectures in the same design, to allow...
comparison of alternatives. The development of best-practices for Constituent Systems engineers can improve the quality and reduce the cost of simulation, by promoting the use of event-based modelling.

### 4.4.7 Future Plans

Future development plans for the DESYRE Joint Simulation tool include the improvement on the tool maturity (including maturation and optimization of the FMU integration algorithm) to reach full industrial strength and the improvement of the user experience by using the feedback collected during the project. Further effort will be devoted to the extension of the tool to support the new version of the FMI standard (2.0).

### 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)

### 4.5.1 Conclusions

While this tool has increased the number of available tools for DANSE users, and one that is connected to the tool-net integration platform, it was not of interest to any of the users and was not used, nor been relevant in fact to any of the use-cases and test-cases. Testing the designs has not been a top priority, nor a goal of DANSE methodology.

### 4.5.2 Lessons Learnt

Obviously, testing that can be, for instance executed over the simulations of the different system components and blocks was neglected in DANSE. However this may be either a mistake, since testing connects requirements to implementations, and that has not been dealt in DANSE at all. Alternatively, it can be considered a result of careful decision of the DANSE proposal to not deal with requirements, nor with testing that this tool could help perform. Therefore, there were no lessons to learn about this tool from the DANSE experience.

### 4.5.3 Future plans

The future plans of the vendor of this tool are not known.
4.6 Contract-based Run-time Verification

4.6.1 Purpose

Within the Synthesis for Diagnosis and Prognosis task, DESYRE Contracts-based Run-time Verification plays a central role. It is integrated in the DESYRE simulation framework [23] and it provides a simulation-based verification tool for goals and contracts specified through the GCSL language. In particular, it allows performing run-time verification based on the contracts foundation, to significantly improve the ability of designers to identify potential negative behaviours both before and after the system deployment.

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. Then, DANSE GCSL contracts are automatically translated into monitor components that continuously interact with the SoS and CS to verify that their inputs and outputs are consistent with the GCSL specification along time. The run-time verification tool automatically generates monitors of GCSL goals and contracts that are linked with the SoS executable model. Contracts monitors produce a Boolean value that becomes false when the contract is violated. It is worth noting that monitor components are generated in the form of FMUs in co-simulation mode, further promoting the usage of the FMI standard in DANSE. As a consequence of this, monitors are stand-alone components that can be also reused and imported in other FMI-compliant simulators.

The Contracts-based Run-time Verification methodology can be seen as a hybrid approach between formal exhaustive (and potentially resource-consuming) exploration and that of simulation where only single traces are sampled and evaluated at each time. The common and unifying point between Contracts-based Run-time Verification and other formal verification techniques in DANSE is the common use of GCSL and its semantics, leading to consistent results across analyses.

The Contracts-based Run-time Verification technique is integrated in the DANSE methodology as it provides a practical method to detect Emergent and Unwanted Behaviours. In particular, the outcome of the analyses can be used to: (1) identify new potential SoS needs, (2) trigger the evaluation of possible architectural changes, (3) identify the most promising SoS architectural/behavioural improvement. Therefore this analysis contributes to the realization of DANSE Capability Learning Cycle as well as to the DANSE Correct by Evolution approach.

4.6.3 Demonstration scenario

Contracts-based Run-time Verification has been applied successfully to the AGT industrial test case (Carmeq) as well as to the CAE [38]. The novelty introduced within DANSE is the support for the analysis of hierarchical systems, where an SoS aggregates several CSs, each having its own behaviour as well as GCSL requirements. GCSL requirements may be in “conflict”, meaning that their composition may be
unrealizable. In particular, we have identified vertical conflicts, where the satisfaction of the composition of CSs requirements does not entail the satisfaction of the SoS requirements (known as dominance test in the theory of contracts), and horizontal conflicts, where the composition of SoS-level or CS-level requirements may lead to the absence of possible behaviours and so to unrealizable integration (known as compatibility or consistency tests in the theory of contracts). The identification and resolution of these conflicts can lead to an improvement of requirements quality and, therefore, to a better SoS design. For the AGT, four verification scenarios were designed:

1) Analysis of horizontal conflicts at CS-level: Traffic Controller goal to maximize traffic flow is compared with City Region goal to minimize local traffic load.
2) Analysis of horizontal conflicts at SoS-level: SoS goal of balancing traffic between controllers is compared with SoS goal to ensure correct load per city region.
3) Analysis of vertical conflicts between SoS-level and CS-level: SoS goal of balancing traffic between controllers is compared with City Region goal to minimize local traffic load.
4) Identification of emergent behaviour: Run-time Verification is used to verify whether the decentralized (i.e. distributed) control architecture realizes the global traffic balancing and flow requirements or not.

**Figure 4-7: GCSL in DANSE**

### 4.6.4 Conclusions

DESYRE Contracts-based Run-time Verification provided a practical tool for combining formal analysis with simulation-based analysis. This proved to be an advantage since it allowed to consider detailed FMU-based behavioural models that would not be suitable for fully-formal analysis due to unmanageable state space
size. Furthermore, monitors generated during the analysis can be further reused for real SoS monitoring during operation.

4.6.5 Lessons Learnt

Monitoring is a well understood technique as it is based on simulation, which is already widespread in industry. The definition of GCSL contracts revealed to be well accepted by the users, thanks to the fact that the GCSL logic is based on natural-language like patterns. However, the understanding and use of the theory of contracts (and, in particular, of the DANSE notion of conflict) seemed to pose more challenges for the full acceptance and industrial use. This means that contracts definition and conflicts identification should be better integrated in the DANSE methodology and adapted to the industrial design flow.

4.6.6 Future Plans

Future plans include bringing the DESYRE Contracts-based Run-time Verification technology to a higher maturity level and applying it to other industrial test cases. The expected outcome of these activities would be an improved integration of the Contracts-based requirements definition methodology into existing industrial design flows. Further plans include the improvement of usability of CS-level GCSL monitors specification. Currently, these should be specified for each CS instance and starting from the SoS root to provide fully-qualified paths to CS attributes. However, this process could be automated by specifying non fully-qualified paths and letting the monitor synthesis tool recover all instances and resolve paths into fully qualified ones.

4.7 GCSL Contracts Analysis

4.7.1 Purpose

The GCSL specification language allow to express: (1) architectural requirements, (2) performance goals, and (3) behavioural contracts. Architectural requirements are used to provide constraints over the possible SoS architectures. Goals express objective functions that should be maximized/minimized to achieve SoS optimal performance. Contracts are used to express constraints on systems behaviour that must be satisfied under any situation. It is important to notice that, due to the independence of the single Systems, both the System of Systems and each Constituent System may come with their own, possibly conflicting, Goals and Contracts. Conflicting Goals may lead to a sub-optimal SoS behaviour, while conflicting Contracts lead to a wrong and potentially harmful SoS behaviour.

The GCSL Contract Analysis technique provides 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 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 DESYRE Contracts Analysis tool covers the fragment of GCSL including time and OCL but not probability, and it is integrated within DESYRE [23] as a plugin. Its use flow is very similar to 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. The analysis problem parameters are grounded using the user settings and the problem is translated into a set of algebraic and Boolean formulas to be resolved by an SMT solver. Behaviors that are admissible by the contracts and demonstrate that two or more of the requirements are in conflict are known as counterexamples. Counterexamples generated by Contracts Analysis can be effectively used to update either the architecture or the contract specification of the system of systems (or both), in order to correct the problem before any further refinement of the SoS/CS behavior is attempted.

### 4.7.3 Demonstration scenario

The GCSL Contracts Analysis tool has been applied to 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 analyze conflicts between the timing allocation of Constituent Systems activities and the timing allocation of System of Systems activities. This is formulated in terms of Dominance between Constituent Systems Contracts and SoS Contracts. In particular, it has been considered the handling of an emergency scenario, where a Pipe Burst event can hinder the quality of the water supply service, if not handled with correct timing. This analysis has been applied at an early design stage where only the SoS BDD diagram was available and the Constituent Systems had neither an FMU-defined behavior nor a net-list was defined for their interconnection (so no UPDM IBD diagram was available). Therefore, from a set of available activity diagrams (UPDM OV-5b diagrams), a number of GCSL contracts were formalized to describe timing requirements for the various Pipe Burst handling and repair activities. Here, in lack of an explicit net-list, the events were considered to be visible from any Constituent System, as if they were broadcasted. A scenario where all the Constituent Systems requirements are fulfilled but the SoS requirement is violated was produced, thus identifying a conflict. The conditions under which such a conflict scenario is realizable were obtained in terms of a timed system events trace, which was used to identify possible reasons for this violation and to establish that the integration of the NationalNetwork and C4I components do not allow to meet the SoS goals. As a consequence of this result the designer considered to revise the GCSL requirements before starting to design the detailed behavior of the Constituent Systems. After revision of the requirements, no further violation was found. Therefore, allowing the designer to proceed in the definition of the detailed behavior of the Constituent Systems with increased confidence in quality of the timing requirements.
4.7.4 Conclusions

Desyre Contracts Analysis provided support for establishing correctness of architecture and contracts attached over it. This information is of crucial importance because it enables the reliable composition of Constituent Systems into a satisfactory SoS. Counterexamples generated by Contracts Analysis can be effectively used to update either the architecture or the contract specification of the system of systems (or both), in order to correct the problem before any further refinement of the SoS/CS behavior is attempted. This tool came late in the third year of the project and was therefore applied only to a limited extent. However it succeeded in proving the feasibility of requirements quality analysis and demonstrating its integration in the DANSE SoS design methodology.

4.7.5 Lessons Learnt

The importance of quality requirements is often underestimated in industry. To increase awareness of this need it is necessary to provide reliable tools and a clear methodology that integrates well with common design flows adopted in industry.
4.7.6 Future Plans

Future plans include bringing the DESYRE Contracts Analysis technology to a higher maturity level and applying it to other industrial test cases. The expected outcome of these activities would be an improved integration of Contracts Analysis into existing industrial requirements definition flows.

4.8 Fault-based Test Generation

4.8.1 Purpose

The purpose of fault-based test case generation is to explore inputs and fault conditions leading to the realization of SoS-level undesirable behaviours, called Top Level Events (TLE). The fault-based test case generation methodology considers simplified SoS/CSs white-box behavioural models including failure and recovery modes. These models are used to produce a set of input traces that exhibit TLEs, made possible by the fact that root cause analysis is performed, starting from single failures, to identify top-level failures. TLEs are formalized by using GCSL requirements. The set of input traces produced by fault-based test case generation can be used in further simulation phases to test the actual occurrence on more detailed models of unwanted SoS behaviours in presence of faults. The DANSE Fault-based Test-case Generation technique can be seen as an extension of fault-based test generation and white-box test-case generation to the case of SoS and with the aim of increasing the SoS designer confidence about robustness to the occurrence of Constituent Systems faults.

The SoS UPDM architecture and the Constituent Systems (white-box) behavioural models, designed using Rhapsody flowcharts, are extended to incorporate the modelling of the error behaviour of the SoS and the CSs. This model should contain a description of the faults that can affect the Constituent Systems and the effects of those faults on their behaviour. The error model may be further elaborated including the fault detection and isolation mechanisms as well as recovery procedures. The generated fault scenarios are exploited to produce fault-injection based test cases that can be used 1) to exercise a fault-tolerant model of the SoS or 2) as a driver for the effective robustness test of a new CS under development.

The hazardous behaviour is formalized as the dissatisfaction of a GCSL contract and typically (but not necessarily) targets the SoS level rather than a CS level. For instance it can be related to the detection of an emergent behaviour: this can be a way to generate a relevant test case, which can be replayed on the real SoS to trigger specifically this emergent behaviour. Such a contract (named Top Level Event) describes a safety property which the SoS should respect.

4.8.2 Tools and Environment

The inputs of the DESYRE Automated Test-case Generation tool are: (1) the SoS architecture specified as an UPDM SV-1 diagram, (2) the nominal (fault-free) Constituent Systems white-box behavioural models specified as UPDM flowcharts enriched with the set of faults also specified as in terms of UPDM flowchart.
states and transations and (3) the unwanted SoS TLE specified as a (set of) GCCL contract(s). The fault-based test-case generation tool is integrated as a plugin of DESYRE.

For each unwanted SoS behaviour the tool generates a test suite, which is a set of simulation traces. Such traces are used to exercise the SoS fault detection mechanisms to verify (by simulation) the SoS tolerance to the modelled faults. The produced set of test scenarios corresponds to the minimum number of simulation executions to have the maximum coverage of the SoS behaviour. To rule out trivial executions, the ATG engine enforces the total number of fault event activations in all possible executions to be as small as possible.

Each generated trace exposes an admissible scenario of the system leading to a hazard condition. This information may be used for analysing the cause and the effects the events chain identifying the root cause of the hazardous behaviour that may arise for different causes such as: a combination of faults of a single CS not properly taken into account by the designers, or an error in the design of the control of the single CS or of the interaction between multiple CSs, or an unsafe emergent behaviour of the SoS exposed after the integration of existing systems or for specific interconnection architectures.

### 4.8.3 Demonstration scenario

A simplified version of the CAE [38] use case, enriched to support fault models and SoS unwanted behaviour models has been used as a demonstration scenario. For this test case a hazard condition related to the capability of the SoS to react on a fire event was identified. The condition in natural language states that “Every time there is a fire, there is at least one active fire-fighting car that is dispatched to the fire area”. The CAE-ATG model violates the TLE above whenever there FireStation is in emergency state and the FireFightingCar state is not active to extinguish the fire. The outputs of the ATG analysis obtained for this example allowed to identify fault injections that lead the FireFightingCar to a Failure state.

The CAE model, enriched with fault-scenarios demonstrated also the use of different levels of abstraction in system modelling. For instance some behavioural models of CSs were translated at a higher level of abstraction. At the same time, a refinement of some parts of the original SoS model were required to support the modelling of the specific TLE. A similar process was done for failure modes, which were not described in the original CAE model. For example, a communication failure state was inserted to the FireStation state chart diagram with two events that trigger the transition to the failure state and another event to trigger the recovery from failure.
4.8.4 Conclusions
The DESYRE Automated Test-case Generation tool and related fault-scenarios generation methodology was applied to identify hazard scenarios and produce fault injections for simulation. This proved to be a valuable activity to provide insights on SoS reliability to the designer.

4.8.5 Lessons Learnt
Explicit modelling of fault behaviours and their effects is a demanding activity as it requires an extra modelling effort and ad-hoc expertise. Furthermore, the complexity of fault modelling grows with the size of the nominal flowchart behaviours. Adoption in industry can be promoted and facilitated by developing a compositional approach to fault scenarios modelling. In the foreseen design flow, the engineer starts by modelling the nominal system behaviour while the specification of fault behaviours should be supported by an alternative UPDM view, not influencing the main, nominal one. This view specifying fault behaviours is, then, taken into account only on need.

4.8.6 Future Plans
Future plans include bringing the DESYRE Automated Test-case Generation tool to a higher maturity level and applying it to other industrial test cases. The expected outcome of these activities would be an improved integration of fault-scenarios generation into existing industrial design flows, including the previously mentioned compositional approach to fault behaviours specification.
4.9 Timing analysis

4.9.1 Purpose

The timing analysis tool (RTANA) 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

![Figure 4-10: Integration of realtimeAnalyzer](image-url)
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 is 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.

### 4.9.3 Demonstration scenario

A simplified version of the CAE model extended by timing annotations and the AGT use case were applied. For the CAE, the Concise tool first determined a set of optimal architectures with respect to operational and static resource costs. These architectures were then verified against a set of timing requirements by the application of RTANA. For the AGT, a pedestrian crossing scenario with both local and global timing requirements was created and validated.

### 4.9.4 Conclusions

The timing analysis technique provides a verification tool for GCSL contracts at Constituent Systems (CS) level and SoS level. The approach works in an iterative fashion enabling to check end-to-end latencies between several CSs and local deadlines of functions on resources of single CSs in a more efficient manner than a holistic analysis. This is because only relevant parts of the state space are kept and interfaces between CSs are minimized. The benefit of using this tool is an exhaustive analysis with respect to the specified timing properties. If there is a faulty timing behaviour between CSs it will be detected by this tool, which would be not detected by a simulation method.

We applied our analysis approach to the CAE, where we analysed architectures obtained from the architecture optimization tool Concise. We also applied our approach to the AGT test case scenario analysing timing constraints concerning communications between CSs. Further, we compared the results of our analysis approach with the state-of-the-art tool UPPAAL considering some smaller examples [55].

### 4.9.5 Lessons Learnt

Although we achieved great progress in the scalability of our analysis, for large architectures scalability remains a problem. Thus, in future projects we extend our minimization approaches and find new abstraction techniques to handle this issue.

### 4.9.6 Future Plans

We will extend our prototype implementation for timing analysis. Specifically, we will investigate new abstraction techniques which will yield more accurate results than the classical analysis techniques and will boost the scalability of our approach. The applied case-studies and several discussions have shown that there is a great demand and still a potential in the scalability of our approach.
Another topic will be the extension of our approach to realize a multi-aspect analysis such that besides timing safety and functional properties can also be verified in a more compositional manner. Further, we will extend our approach with an impact analysis approach. First results concerning this topic were already presented in the RTCSA conference [54]. With this approach, we are able to avoid re-validations of parts of the SoS both specification and on implementation level, which did only change in a "good manner".
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 Rhapspdy 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 lugin 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.

3. Exporting to the server.

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

Figure 5-7: Apply the SMC import command

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.
5.1.4 Enhancement in Final Prototypes (including III)

UPDM capabilities have been added and demonstrated, 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.

The following figures show two screens of an CAE model in UPDM first, than as it is imported into a SysML model on Rhapsody following the mediation step.
Figure 5-11: UPDM model in the Rhapsody Explorer that is exported.
5.1.5 Tool development

This tool has been rapidly developed throughout the DANSE project, as in the revisions log in the following list:

- Jan 14 2015, v1.28 - fixed bug reported from SODIUS.
- Dec 15 2014, v1.28 - MDW mediator now works.
- Nov 17 2014, v1.27 - More info when an interceptor is found invalid.
- Oct 19 2014, v1.27 - Showing port type icons, and options to build any port type from ontology.
- Sep 08 2014, v1.26 - Resolved some UI bugs.
- Jul 13 2014, v1.24- SPARQL UI improved
- Jul 2 2014, v1.24- Graphing of SPARQL queries
- Jun 30 2014, v1.23- More SPARQL queries management
- Jun 23 2014, v1.23- SPARQLs expanded to full syntax
- Jun 16 2014, v1.23- SPARQLs can be saved with names and reused
- Jun 12 2014, v1.22- Import from origin, and not the intermediary friends.
- Jun 06 2014, v1.21- Protect imported ontologies from modifications, load and protege editing.
- Jun 02 2014, v1.21- Extended distribution: can share ontologies and rules among SMC servers.
- May 22 2014, v1.20- Extended automation to ontologies and rules, incl testing of direct repositories.
- May 18 2014, v1.19- cleaned GUI problems
- Apr 27 2014, v1.19- Improved sparql option 'Used By' + 'look back' in the resource RDF browsing.
- Feb 23 2014, v1.18- Mediation view shows associated resources inline.
- Feb 18 2014, v1.18- Ontologies governance: distinguishing import from upload.
- Feb 13 2014, v1.18- Export ports inform better on their validation; Large models now work in pages for the table view option.
- Jan 26 2014, v1.18- Ontologies/repositories graphs show blank nodes properly now + configuring rul-sets view + unified style of tables.
- Jan 23 2014, v1.18- L&F with tabs + Uploading of ontologies now support multiple formats + Ontology compare fixed a bug.
- Jan 7 2014, v1.16- Clean start after new installation + buttons + adoption for projects in progress.
- Dec 20 2013, v1.15 - Graph view of repositories, using graphviz.
- Dec 05 2013, v1.14 - DM compliant version - can work with dm/web projects + Fixed bug in configuration migration.
- Dec 02 2013, v1.13 - Repository comparison improved. Byte coding of textual fields is now fully respected.
- Nov 17 2013, v1.13 - OSLC AM Service Provider comes to life. TBD. Follow this new Icon: 🌟

<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>2015-02-15</td>
<td>96</td>
</tr>
</tbody>
</table>
- Oct 8, 13: v1.9 - Bug fix in oslc GET.
- Sep 23, 13: More bugs in configuration UI fixed.
- Sep 16, 13: Some bug fixes and performance improvements.
- Aug 8, 13: New revision 1.6: Distributed SMC introducing Friends. CSS styles used for L&F.
- Jul 11, 13: Customization possible for list mode browsing of repositories.
- Jun 23, 13: Added Jazz Look and feel style, only for RTC widget services.
- May 28, 13: Added license "click through" service for mediators.
- May 7, 13: Integrated online Protege ontology editing.
- Apr 14, 13: Repository bug fixed.
- Apr 9, 13: Improved repository model RDF browsing.
- Mar 20, 13: Clear all filters button + some internal refactoring.
- Mar 18, 13: Fixed bug in repository comparisons.
- Mar 13, 13: Downloads available for ontologies and repositories RDF contents.
- Mar 11, 13: Blob support with permanent attachment repository using a permanent built-in SM ontology.
- Feb 28, 13: Item editing filters candidates according to roles and tags.
- Feb 19, 13: Mediator extension points problem fixed. Comparisons of repositories improved with hot-points.
- Feb 13, 13: Mediators config now specify interceptors from a list of names in the plugin extension point SmMediator.
- Feb 3, 13: Added Widget oriented view of repositories (action=ShowShort)
- Jan 21, 13: Mediators now keep links and can show them.
- Jan 8, 13: Item tagging working to filter display.
- Jan 8, 13: This limitation obsolete: Container flows should be limited to single stages: Tools Repositories.
- Jan 1, 13: Batch (no-wait) mediation; Buttons reorganized; Graphics are persisted.
- Nov 21, 12: Full recursive genuine status updates.
- Nov 20, 12: Show port access names in refs. Indicates if missing ports/ontology/rules in references.
- Oct 30, 12: Old separate repositories view removed. Use repository Ports for that.
- Oct 21, 12: Improved support for SPARQL and OSLC in repositories display.
- Oct 15, 12: Better information in errors reporting.
- Oct 6, 12: Repositories resources URL now use / instead of #.
- Oct 6, 12: Collapse/Expand works!, Graphics buttons work too! Using meaningful names.
5.1.6 Conclusions

The semantic mediation container had a mixed reaction from the users. On the positive side, AirBus developed the CAE and had a keen interest in this technology to automate the manual work done during the initial stages of that experiment in applying optimization. That has also been the driver behind the UPDM to SysML mediation which both demonstrated usefulness of semantic mediation, as well as improving the work process and making some positive points for the DANSE methodologies.

Other industrial users have enjoyed these capabilities only based on needs, and depending on the other tools needed for their work. So is the work on AGT in which simulation by DESYRE enjoyed a great level of automation once it has been connected with the tool-net semantic mediation container, and that has also been demonstrated.

An important value of this platform is in breaking the borders among tools and creating a great opportunity for reuse of models. The Architecture patterns are an excellent example of that opportunity. However, there was not enough drive and available effort to integrate the catalogue of patterns to the tool-net. That is a miss of opportunity (see also future plans).

To make tools integration a reality, it seems – at least by some of the partners – important to show how easy it is to integrate new tools for both vendors, and also for users who will appreciate more the opportunities in tools integration.

5.1.7 Lessons Learnt

Integration is secondary to most users over tools functionality. Users are eager to create new value from new tools and care less on how to save on the repeated work that starting up with such tools requires. That can be also considered a problem with MBSE as a whole, where modeling tools are hard to learn. The tool-net integration in DANSE was easier to view as a not-integrated collection of tools, where the effort to learn and use these tool seem to be enough over the value of integration - both to the industrial partners, and to the vendors.

A focus on tools integration in the test cases may have shifted some of the efforts from the exercise in SoS design alone, to also a research in tools integration.

Need to also make it clearer the ease of adding new tools into the integration environment and platform.

5.1.8 Future Plans

The Tool-net integration is a promising technology and will be pursued. Partners in DANSE have interest to continue working on that also beyond the project timeline, investigating the possibilities and publishing new work as we progress. The enablement of model reuse through this tool-agnostic approach is something to look for.
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.

![Figure 5-13: Installing Protege 4.3.](image)

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.

![Figure 5-14: Control panel of the SMC plugin to Protege](image)
The Protégé editing desktop will reflect OWL ontologies of SMC as in the next figure:

![Protégé editing session with an ontology from SMC.](image)

**Figure 5-15:** Protégé editing session with an ontology from SMC.

### 5.2.1 Conclusions

As a tool for the “Power User” this tool was not used by the industrial users, but the addition of tools to the integration platform required to work on ontologies and rules. Protégé is no doubt the best tool for that and integrating it helps a lot.

Having said that, there are also more manual methods to do the same, yet in favor of the automation principle that DANSE advocates, the integration makes an important step to ease the work needed to add new tools to the integrated tool box.

Caveats are that rules-editing and ontologies for semantic mediation have a bit different needs than this general tool.

### 5.2.2 Lessons Learnt

A good tool for its purpose, yet as a tool for mediation both for ontologies and for rules, a different tool would be needed, perhaps in combination with this tool as well.

### 5.2.3 Future Plan

Use this integration to promote the addition of new tools to the semantic mediation integration approach. When learning better the special needs of this approach – go beyond this tool for more specialized solutions.
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 5-16: Example of MDWorkbench documentation](image)
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.

![My First Mediator](Image)

**Figure 5-17:** A page from MDWorkbench semantic mediator tutorial

### 5.3.3 Conclusions

The whole content of Sodius solutions: Tool-net clients, metamodels, ontologies, semantic mediators, were developed by MDWorkbench mediation rules editor and its Eclipse platform. It confirms the viability of the product for extending the Tool-net in all aspects of model sharing. Complete documentation and tools were created for helping developer at each step of his work, as guides, tutorials, and helpers for providing materials in a format respecting Tool-net requirements. And finally a tutorial shows all the process to follow for extending the Tool-net from the definition of a new ontology to the usage of Rhapsody Client with a custom semantic mediator.

### 5.3.4 Lessons Learnt

Documentation and tutorials are the main entry point for any developer interested about semantic mediation development. The richness of functionalities can repel users if they are not guided gently at each problematic step.

### 5.3.5 Future Plans

The integration of several automated tool could drastically improve semantic mediation development experience.
5.4 SMC client SDK

Note: No further changes in Prototype III and beyond.

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 use 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:

![Sample Panel (import/export, non-modal)](image)

**Figure 5-18:** Example SMC client to demonstrate the SDK.

Being an SDK, this tool comes with a detailed javadoc documentation of the API.
Figure 5-19: Javadoc documenting the SMC client API.

5.4.1 Conclusions

This package is very useful in enabling new tools. Experiment in DANSE shows it has been easy to use for the vendors users role of the tool-net integration platform.

5.4.2 Lessons Learnt

A must-be capabilities to add more tools to the tool-net integration.

5.4.3 Future Plans

Adopt to support new SMC platforms hand in hand when making these available so tools can migrate to work with the new platforms. E.g., none-Jazz components.
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.

At the end – this tool has not been developed.

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.

At the end – this tool has not been developed.
6 Summary

The final DANSE prototype is presented here as the culmination of 3 prototypes and a final period of application by the users. With the reactions of users, some enhancements and fixes in the tools brought them to a higher maturity than in earlier prototypes. Some of the additional technologies added in the 3rd prototype, as well as some earlier ones, seem to not gain sufficient value to the users and were not applied in the test cases.

Integration of tools with the tool-net and the semantic-mediation container have also progressed to the last moment of delivery of this document.

A final picture of the tools usage and value to the partners has been gained by concluding this project, and some of that has been reflected in conclusion and lessons learnt reported at each of the different tools above.

6.1 Conclusions

Some of the conclusions on the prototype have been drawn when reporting on the Tool-Net Semantic Mediation Container (SMC) platform. Tools integration has not been a prime goal of the project, and as the project has unfold to test cases by the industrial partners, it remained secondary.

Having said that, the CAE example that has been explored throughout the project, showed stronger interest in the integration and the promise that this can provide to a user working in MBSE and interested to push on with the reuse of models. Reuse seems to be an important point in justifying MBSE. That is a point in case for AirBus (who lead the CAE) appreciation of the value of tools interoperability and integration.

Tool vendors had a mixed approach, where most had interest to integrate their tools, new and old. SODIUS have provided all their tools through this integration, while OFFIS did not find the means to do that with their tools. LU developed an excellent environment for reuse of model patterns intended for that purpose, but did not prioritize integration enough to do it, although maintaining that this issue is important and in their plans. ALES was able to both develop new tools in their DESYRE suite of tools, and also integrate it into the tool net as a secondary effort which reached some maturity level towards the end of the project. Thanks to that and the importance of the DESYRE tools to the test cases, we can demonstrate important automation flows through the tools-net.

6.2 Lessons Learnt

Tools integration should be a declared prime goal of a project to make it successfully applied. Without the interest of the users this will not happen. Vendors have clear interest in integration, yet when the project is driven by use-cases, where integration and automation are not prime goals – they too can be drawn to de-prioritizing integration over functionality. Each of the different technologies has drawn its own lessons and plans from this project. Some of that is reported in the chapters above.
6.3 Future Plans

See specifics above per each technology and tool in this prototype.
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>Abbreviation</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 Elements.</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

**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&quot;</td>
</tr>
<tr>
<td>Reference</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>[22]</td>
<td>GROOVE: GRaphs for Object-Oriented VErification (GROOVE), <a href="http://groove.sourceforge.net/groove-index.html">http://groove.sourceforge.net/groove-index.html</a></td>
</tr>
<tr>
<td>[28]</td>
<td>DM: Jazz Design Management <a href="https://jazz.net/products/design-management/">https://jazz.net/products/design-management/</a></td>
</tr>
<tr>
<td>[31]</td>
<td>D8.2.3: Conceptual and architecture principles of SoS design and semantic interoperability of systems platform and SoS design Tool-Net, DANSE delivery for WP8 on M17.</td>
</tr>
<tr>
<td>[33]</td>
<td>Modelica: <a href="https://www.modelica.org/">https://www.modelica.org/</a></td>
</tr>
<tr>
<td>[34]</td>
<td>Dymola: <a href="http://www.3ds.com/products/catia/portfolio/dymola">http://www.3ds.com/products/catia/portfolio/dymola</a></td>
</tr>
<tr>
<td>[38]</td>
<td>CAE: “D3.3: Concept alignment example description,” DANSE delivery for WP3 on M12.</td>
</tr>
<tr>
<td>[40]</td>
<td>D6.5.1: “D6.5.1: Extension of standard profiles for DANSE Modeling”.</td>
</tr>
</tbody>
</table>
programming, mixed integer programming, and quadratic programming," By ILOG or IBM: [link](http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/)

<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[42]</td>
<td>Prototype-I DANSE deliverable D8.3 – Prototype I.</td>
</tr>
<tr>
<td>[43]</td>
<td>D3.6 DANSE deliverable D3.6 – Validation Report - Prototype Iteration I</td>
</tr>
<tr>
<td>[47]</td>
<td>D3.7 DANSE deliverable D3.7 – Validation Report – Prototype II</td>
</tr>
<tr>
<td>[48]</td>
<td>D4.3 DANSE deliverable D4.3 – DANSE Methodology V02.</td>
</tr>
<tr>
<td>[50]</td>
<td>D4.4 DANSE deliverable D4.4 – DANSE Methodology V03.</td>
</tr>
<tr>
<td>[51]</td>
<td>D3.8 DANSE deliverable D3.8 – DANSE Validation Report</td>
</tr>
<tr>
<td>[52]</td>
<td>D8.5 DANSE deliverable D8.5 – DANSE Prototype III</td>
</tr>
<tr>
<td>[53]</td>
<td>D8.7.3 DANSE deliverable D8.7.3 – DANSE Exploitation Forum V3.</td>
</tr>
</tbody>
</table>