Designing for adaptability and evolution in system of systems engineering

Prototype I

D_8.3

<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>Leonardo Mangeruca</td>
<td>ALES</td>
<td><a href="mailto:leonardo.mangeruca@ales.eu.com">leonardo.mangeruca@ales.eu.com</a></td>
</tr>
<tr>
<td>Benoit Boyer</td>
<td>INRIA</td>
<td><a href="mailto:Benoit.Boyer@inria.fr">Benoit.Boyer@inria.fr</a></td>
</tr>
<tr>
<td>Evegeny Shindin</td>
<td>IBM</td>
<td><a href="mailto:EVGENSH@il.ibm.com">EVGENSH@il.ibm.com</a></td>
</tr>
<tr>
<td>Roy Kalawsky</td>
<td>LU</td>
<td><a href="mailto:R.S.Kalawsky@lboro.ac.uk">R.S.Kalawsky@lboro.ac.uk</a></td>
</tr>
<tr>
<td>Yann Loubopin</td>
<td>SODIUS</td>
<td><a href="mailto:ylebeaupin@sodius.com">ylebeaupin@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>2013-03-21</td>
<td>Initialized</td>
<td>All</td>
</tr>
<tr>
<td>0.2</td>
<td>2013-4-12</td>
<td>Added Graph Grammar section</td>
<td>9</td>
</tr>
<tr>
<td>0.3</td>
<td>2013-4-22</td>
<td>Added Simulation from ALES</td>
<td>5, abbreviations</td>
</tr>
<tr>
<td>0.4</td>
<td>2013-4-23</td>
<td>Added Statistical Analysis</td>
<td>6, abbreviations</td>
</tr>
<tr>
<td>0.5</td>
<td>2013-4-24</td>
<td>Optimization added + scenario for simulation + FMI</td>
<td>Renumbered, 2, 4, 6, abbrev, references</td>
</tr>
<tr>
<td>0.6</td>
<td>2013-4-29</td>
<td>Architecture Patterns</td>
<td>Chapters 8, 11</td>
</tr>
<tr>
<td>1.0</td>
<td>2013-4-30</td>
<td>UPDM</td>
<td>Chapter 3.</td>
</tr>
<tr>
<td>1.0</td>
<td>2013-5-3</td>
<td>Architecture patterns scenario update from Roy</td>
<td>Chapter 8.3</td>
</tr>
<tr>
<td>1.0</td>
<td>2013-5-5</td>
<td>Graph Grammar scenario update from Christoph</td>
<td>Chapter 7.3</td>
</tr>
</tbody>
</table>
CONTENT

1 OVERVIEW, PURPOSE AND SCOPE ........................................................................................................ 8

2 SEMANTIC MEDIATION ................................................................................................................................ 10

2.1 PURPOSE ............................................................................................................................................... 10
2.2 TOOLS AND ENVIRONMENT ................................................................................................................... 10
  2.2.1 Jazz/DM ............................................................................................................................................. 10
  2.2.2 Rhapsody ........................................................................................................................................... 11
2.3 DEMONSTRATION SCENARIO ................................................................................................................. 11

3 UPDM/NAF ..................................................................................................................................................... 14

3.1 PURPOSE ............................................................................................................................................... 14
3.2 TOOLS AND ENVIRONMENT ................................................................................................................... 14
3.3 DEMONSTRATION SCENARIO ................................................................................................................. 16

4 CONCISE MODELING .................................................................................................................................. 19

4.1 PURPOSE ............................................................................................................................................... 19
4.2 TOOLS AND ENVIRONMENT ................................................................................................................... 19
4.3 DEMONSTRATION SCENARIO ................................................................................................................. 20

5 SIMULATION .................................................................................................................................................. 23

5.1 PURPOSE ............................................................................................................................................... 23
5.2 TOOLS AND ENVIRONMENT ................................................................................................................... 23
  5.2.1 FMU generation on Rhapsody ............................................................................................................. 25
5.3 DEMONSTRATION SCENARIO ................................................................................................................. 25

6 STATISTICAL ANALYSIS .............................................................................................................................. 27

6.1 PURPOSE ............................................................................................................................................... 27
6.2 TOOLS AND ENVIRONMENT ................................................................................................................... 27
6.3 DEMONSTRATION SCENARIO ................................................................................................................. 28

7 GRAPH GRAMMARS ...................................................................................................................................... 29

7.1 PURPOSE ............................................................................................................................................... 29
7.2 TOOLS AND ENVIRONMENT ................................................................................................................... 29
7.3 DEMONSTRATION SCENARIO ................................................................................................................. 33

8 ARCHITECTURE PATTERNS .......................................................................................................................... 37

<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>2013-05-05</td>
<td>4</td>
</tr>
</tbody>
</table>
8.1 PURPOSE ................................................................................................................................. 37
8.2 TOOLS AND ENVIRONMENT .................................................................................................. 39
  8.2.1 SOS Architecture Pattern Repository ............................................................................. 39
8.3 DEMONSTRATION SCENARIO ............................................................................................... 42
9 ABBREVIATIONS AND DEFINITIONS .................................................................................... 43
10 REFERENCES ......................................................................................................................... 48
11 APPENDIX A – ARCHITECTURE PATTERNS ...................................................................... 51
  11.1 SOS ARCHITECTURE PATTERN REPOSITORY ................................................................. 51
  11.2 DOCUMENT ....................................................................................................................... 51
  11.3 RHAPSODY MODELS ........................................................................................................ 51
  11.4 PATTERNS ONTOLOGY ..................................................................................................... 52
Content of Figures

Figure 1-1: Tools net for the DANSE prototypes .................................................................8
Figure 2-1: Installing the Rhapsody plugin for semantic mediation export/import SysML models .................................................................11
Figure 2-2: Applying this plugin as as profile to a project. ....................................................11
Figure 2-3: Activate the export command ...........................................................................12
Figure 2-4: Setting up user credentials and target server. ..........................................................12
Figure 2-5: Successful export done ....................................................................................12
Figure 2-6: Create a new SysML project and apply to it the plugin sm-dm-2.1. .......................12
Figure 2-7: Apply the import command ................................................................................12
Figure 2-8: Enter user credentials, source server IP and URI of an element to import...............13
Figure 2-9: Populate a diagram from the created model for this import. ................................13
Figure 2-10: End result of this import. ...............................................................................13
Figure 3-1: UPDM Mediation MDWorkbench Support ..........................................................14
Figure 3-2: MOF Profile/Metamodel to OWL Ontology “Productivity” Tool Chain ..................15
Figure 3-3: UPDM Mediation instantion into the Jazz/DM Tool-Net ........................................15
Figure 3-4: UPDM Sharing – Preparation and Initial State ....................................................16
Figure 3-5: MDWorkbench UPDM Data and Diagrams Storage .............................................17
Figure 3-6: MDWorkbench System Architect Tool-Net Connection ......................................17
Figure 3-7: MDWorkbench Rhapsody Tool-Net Connection ................................................17
Figure 3-8: MDWorkbench Rhapsody Data and Diagram Import .........................................17
Figure 3-9: Tool-Net Constituents Sharing at different depth levels .......................................18
Figure 4-1: SoS optimization workflow ..............................................................................20
Figure 4-2: Communication system technical internal block diagram .................................21
Figure 4-3: Optimal communication system architecture. ....................................................22
Figure 5-1: DANSE System of System Design & Simulation Workflow ...............................23
Figure 5-2: Tool-net integration for DESYRE .....................................................................24
Figure 6-1: Statistical Model Checking tool chain in DANSE .............................................27
Figure 7-1: UPDM model of the CAE ...............................................................................30
Figure 7-2: GROOVE model ............................................................................................30
Figure 7-3: Network of architectures ...................................................................................31
Figure 7-4: Invariant rule ....................................................................................................32
Figure 7-5: Evolution with invalid architectures ...................................................................32
Figure 7-6: UPDM StV-6 view (taken from the UPDM 2.0 specification document) ..................33
Figure 7-7: Rhapsody UPDM model ..................................................................................33
Figure 7-8: Rule for one local change ................................................................................34
Figure 7-9: Represents Tag in a Placeholder .........................................................................35
Figure 7-10: Final Rule ......................................................................................................35
Figure 8-1: Illustration showing applicability of architecture patterns over the evolution of a SoS ...............38
Figure 8-2: SoS Architecture Design Space Explorations ........................................................................38
Figure 8-3: SoS Architecture Patterns incorporation within the modeling framework ..........................40
Figure 8-4: Six stage process showing architecture pattern process ....................................................40
Figure 8-5: Exploring Alternative SoS Architectures ..............................................................................41
Figure 8-6: UPDM Views for SoS Description .........................................................................................41
Figure 11-1: Decentralised Command Pattern in Template .......................................................................51
Figure 11-2: Example of Patterns Profile in Rhapsody .............................................................................52
Figure 11-3: Patterns Repositories’ Integration ......................................................................................53

Content of Tables

Table 1-1: Outline of prototype cases described in this document .................................................................9
Table 5-1: Mapping from SysML to FMI .....................................................................................................25
1 Overview, Purpose and Scope

The first DANSE prototype consists of tools and services that partners have worked out during the first 18 months of the project, some of which have been demonstrated in the year 1 review. The prototypes do not yet constitute a continuous network of integrated tools, something which the consortium will aim to achieve during the next 2 iterations. The following Figure 1-1 shows an example of flows among tools in the DANSE tools-net which this prototype demonstrates.

Figure 1-1: Tools net for the DANSE prototypes

The tools interoperability automation is a work in progress and only part of the prototype indeed demonstrate that automation. The following Table 1-1 lists the individual prototype elements and the capabilities they can demonstrate for this M18 of the project. Follow-up prototype will show progress both in capabilities and in integration and automation of the tools interoperability.

Note: This document finalization required collecting of material from all partners and that caused a slight delay of 5 days in the final delivery.
<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Tools</th>
<th>Description</th>
<th>Participants</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Semantic Mediation</td>
<td>Jazz/DM, Rhapsody</td>
<td>Semantic mediation platform on Jazz/DM serving mediation of RDF models between Rhapsody and another Rhapsody based on SysML.</td>
<td>IBM</td>
<td>Delivered</td>
</tr>
<tr>
<td>3</td>
<td>Concise modeling</td>
<td>Rhapsody, MS Excel, CPLEX</td>
<td>Performing a multi-objective parameterized optimization from an architectural pattern in Rhapsody and a list of parameters in MS Excel, using CPLEX solver.</td>
<td>IBM</td>
<td>Delivered</td>
</tr>
<tr>
<td>4</td>
<td>Simulation</td>
<td>DESYRE, Rhapsody</td>
<td>Generating FMU objects for a behavioural mode on Rhapsody, that are used for co-simulation with DESYRE.</td>
<td>IBM, ALES</td>
<td>Delivered</td>
</tr>
<tr>
<td>5</td>
<td>Statistical Analysis</td>
<td>PLASMA, DESYRE</td>
<td>Using DESYRE to perform statistical analysis of a model containing contracts, directly activating the PLASMA analyser for that.</td>
<td>ALES, INRIA</td>
<td>Delivered</td>
</tr>
<tr>
<td>6</td>
<td>Graph grammars</td>
<td>GROOVE, Rhapsody</td>
<td>Using GROOVE to demonstrate the value of graph grammars in changing and creating multiple alternatives to a graph model.</td>
<td>OFFIS</td>
<td>Delivered</td>
</tr>
<tr>
<td>7</td>
<td>Architecture Pattern</td>
<td>Rhapsody</td>
<td>Using SysML profiles as a language to describe SoS architectural patterns.</td>
<td>Loughborough University</td>
<td>Delivered</td>
</tr>
</tbody>
</table>

**Table 1-1:** Outline of prototype cases described in this document
2 Semantic Mediation

2.1 Purpose

Semantic mediation platform on Jazz/DM serving mediation of RDF models between Rhapsody and a common SysML ontology. This SysML ontology is common to a Modelica tool that is not included in the prototype, not being part of the DANSE consortium, although the mediation to an RDF compatible with this tool is included on the platform. A separate mediation for UPDM/NAF is described in Error! Reference source not found. below, all sharing the same interoperability principle and architecture that are described in the D.8.2.3 deliverable [31].

2.2 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.

2.2.1 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 later 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.
2.2.2 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.

2.3 Demonstration scenario

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

![Figure 2-1: Installing the Rhapsody plugin for semantic mediation export/import SysML models.](image1)

![Figure 2-2: 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.
3. Exporting to the server.

4. Import to a new project

Figure 2-3: Activate the export command

Figure 2-4: Setting up user credentials and target server.

Figure 2-5: Successful export done.

Figure 2-6: Create a new SysML project and apply to it the plugin sm-dm-2.1.

Figure 2-7: Apply the import command
Figure 2-8: Enter user credentials, source server IP and URI of an element to import.

Figure 2-9: Populate a diagram from the created model for this import.

Figure 2-10: End result of this import.
3 UPDM/NAF

3.1 Purpose

Using Semantic Mediation as in #1, 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 several RDF/OSLC clients and providers for the Tool-Net. This preliminary work has validated in the Prototype 1 the capability of 3\textsuperscript{rd} parties to enhance DANSE platform and interact deeply with the Tool-Net assets.

![Figure 3-1: UPDM Mediation MDWorkbench Support](image)

3.2 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

---

**Figure 3-3:** UPDM Mediation instantiation into the Jazz/DM Tool-Net
3.3 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 3-4: 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 3-9: Tool-Net Constituents Sharing at different depth levels
4 Concise modeling

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 turn process of finding optimal design to an extremely difficult task. The purpose of concise modeling 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.

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 4-1: SoS optimization workflow. The process can be repeated for different architectural patterns to obtain set of optimal solutions over set of architectural patterns.
4.3 Demonstration scenario

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

- Geographical domain knowledge utilized by existing communication system antennas disposition, possible places for new antennas and maximum numbers of antennas in selected positions.
- Radio-electronic domain knowledge utilized by coverage tables, communication equipment types and possible equipment connections.

The main parts of the potential communication system topology shown in Figure 4-2: 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 4-3: Optimal communication system architecture. This architecture provides 98.5% coverage for both networks (which best possible coverage) by minimal cost.
Figure 4-3: Optimal communication system architecture.
5 Simulation

5.1 Purpose

The purpose of the technology is the simulation of an executable subset of a UPDM [8] model. The executable subset includes internal block diagrams. Block implementations are accepted as URI references to FMU (Functional Mockup Unit) packages that implement the FMI (Functional Mockup Interface) standard [32]. FMU is generated on Rhapsody from SysML projects with state-charts diagrams in blocks.

5.2 Tools and environment

DANSE methodology for System of System modeling and analysis plans to specify Constituent Models using different modeling languages, depending on the characteristics of the system to model and on the actual capabilities and experiences of the system designer. According to the DANSE methodology, the constituent system models are then exported as FMU and published on the DANSE tool-net, ready to be used by different tools. The SoS model is specified as a composition of constituent system models that can be retrieved directly from the tool-net. The Rhapsody UPDM language has been selected as the SoS modeling language. Once the SoS model has been completed, the designer can annotate it with a set of auxiliary information, such as: goals, contracts, stochastic parameters, metrics and so on. This can be done using the set of tags provided by the DANSE profile for UPDM.

Completed the specification and annotation of the SoS model, the designer will be able to use IBM architect to perform some architecture optimization of the SoS and GROOVE [22] to generate possible life-cycle evolutions of the SoS during its life-cycle. For each alternative of the SoS model resulting from both the architecture optimization and the life-cycle evolution evaluation phases, simulation can be used to provide performance analyses and statistical model checking.

Figure 5-1: DANSE System of System Design & Simulation Workflow
In order to simulate the SoS model it is necessary to transform the Rhapsody UPDM model into a format that can be interpreted by DESYRE [23], the simulation framework. A model transformation will then transform the UPDM SoS structure into a DESYRE SLang model while SoS model annotations are collected and used to configure the model, the metrics for performance evaluation and the statistical model checker PLASMA-LAB Error! Reference source not found..

Once the simulator and the analysis tools have been correctly configured, the simulation can be run and simulation traces can be elaborated by the analysis tools providing the user with the expected analysis results.

![Diagram](image)

**Figure 5-2**: Tool-net integration for DESYRE

**Figure 5-2** shows the tool flow for simulation. Models for constituent systems can be designed using several tools, provided the support the export for FMI component integration – such as with Rhapsdoy (see below). Examples of such tools include most Modelica-based [33] tools, such as Dymola [34] and JModelica [35], and Simulink [36]. IBM also provides a prototype of this functionality for Rhapsody [37]. The exported components must be published over the DANSE tool-net. The DANSE extensions to UPDM include the capability of specifying by stereotypes the FMU associated with a UPDM system or component. DESYRE provides a Rhapsody plug-in to publish an executable subset of UPDM to the tool-net. On the DESYRE side the publish UPDM subset is imported into DESYRE from the tool-net. Through the DEYSRE dashboard, the user is able to run several simulation-based analysis. The DESYRE environment collects the results of such analysis and publishes them back to the tool-net.
### 5.2.1 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 5-1):

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

Table 5-1: 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 tool as described above.

### 5.3 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 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 5.2.1 above). 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.
6 Statistical Analysis

6.1 Purpose

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

6.2 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 Figure 5-2. 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 6-1 gives more details about the PLASMA-LAB workflow especially with DESYRE.

![Figure 6-1: Statistical Model Checking tool chain in DANSE](image)
As already explained in Subsection 5.2, the DESYRE collects the compiled model from Rhapsody. Similarly, it also gets the contracts attached to this model. These contracts defined using the Goal Contract Specification Language that completes the UPDM modelling by attaching a formal specification of some goals. This language is more readable and thus easier to understand than LTL [26]. It is closer to the handwritten specification but with a formal semantics whereas LTL is a very low level language and LTL is generally understood by experts only. Each GCSL [24] goal is translated by an external compiler invoked by DESYRE to produce an equivalent LTL formula, the input specification language of PLASMA-LAB. Next, the user can start a SMC [27] session, to start PLASMA-LAB that starts the analysis. This analysis is based on simulations that are provided by DESYRE each time PLASMA-LAB requires new simulation: During the SMC session, DESYRE becomes temporarily the plugin of PLASMA-LAB. When the session finished, PLASMA-LAB returns the results collected by DESYRE.

### 6.3 Demonstration scenario

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

7.1 Purpose

Architectural changes are part of the self-adaptiveness of an SoS. One major challenge is therefore to ensure that such self-adaptive SoS fulfils its global goals and contracts over time. To model the evolution of the SoS from an architectural point of view graph grammars can be used to model only local changes of the architecture. A sequence of those local changes represents one evolution path that the SoS may take. The set of all sequences build all evolution paths a SoS can take. The gain of using the graph grammar approach is to be able to reason about reachable future architectures without exploring all instances of possible architectures for properties related to the architecture (e.g. connectivity of CSs). On the other hand the generated architectures can also be used as input for detailed simulation which would complement the pure architecture with the behaviour of the CS and the environment.

7.2 Tools and environment

The generation of architectural alternatives based on graph grammar rules is implemented by using different software components to create the functionality required for DANSE users. The input graph which is used to apply the grammar rules on is generated from a UPDM model in Rhapsody. As an example see Figure 7-1 which is a screen shot of the Rhapsody model and Figure 7-2 which represents the same model in the graph grammar tool GROOVE [22]. We use the GROOVE tool to generate the alternatives and therefore a transformation step from the Rhapsody UPDM model into the GXL graph format which is used by GROOVE is needed. To feedback the generated architectures to UPDM an inverse transformation will be implemented for the next prototype iteration.
Figure 7-1: UPDM model of the CAE

Figure 7-2: GROOVE model
To enable the user to model not only the SoS model in UPDM/Rhapsody an extension profile delivers the possibility to also model the grammar rules in a UPDM-like style in Rhapsody. A detailed description of this profile can be found in the D6.5.1 deliverable (“Extension of standard profiles for DANSE Modelling”). The UPDM model plus the rules are used to generate a set of architectures and one feature of GROOVE checks for each new architecture if it is isomorphic to an already generated one. Thereby not only a tree of possible evolutions is generated but an directed network. In Figure 7-3 an evolution with several isomorphic architectures is shown. In this case the evolution is quite symmetric and ends for all paths in the same architecture which is of course a special case.

Figure 7-3: Network of architectures

To investigate if unwanted architectures are reachable by the application of the rules to the initial model the certain rules can be annotated as constraint rule. Constraints are pattern which mark an unwanted pattern in the architecture. In Figure 7-4 such an invariant rule is presented and it matches if two FireBrigades operate in the same Region but using different communication technologies that interfere. The rule would, if applied, remove the communication relation to the CCC.
The generator can take such invariants to stop the evaluation of the evolution and thereby invalid architectures and the paths of change are identified. In Figure 7-5 the constraint rule stops the evaluation of each path ending in an invalid architecture. This reduces on the one hand the computation time of the entire evolution but more important it allows identifying all architectures which are on a path to a valid one and when paths to invalid ones are branching. In this case it is not surprising that the choice which pair of FireBrigades is send decides if the invalid states are reachable but this is in general not that easy to identify.

In the first prototype iteration only this use case is supported but for later versions a feedback of the generated architectures will be implemented. This extension will allow to not only check architectures against architectural pattern but also to simulate generated architectures and verify properties which take system dynamics into account. If for example the response time requirement from the CAE should be taken into
account no only the architecture but also behavioural aspects of the CSs, the environment and their interaction must be considered as well.

For the first prototype iteration the user will be able to specify the evolution and constraint rules in Rhapsody and the generated architectures will be only available in the GXL format. This limitation should not prevent the user to do the modelling of the evolution and to get first feedback from the generator which serves to identify future SoS architectures (valid or invalid ones).

7.3 Demonstration scenario

To demonstrate the usage of the tooling we chose a very simple example which can easily be reproduced. The example is derived from the example included in the UPDM 2.0 specification which describes how to apply the UPDM methodology and meta model on an maritime search and rescue SoS.

From the example we took the StV-6 Operational Activity to Capability Mapping view (see Figure 7-6) to illustrate the architecture generation approach.

The left part of the StV-6 view including the Capabilities “Search” and “Inform” as well as the Operational Activities “Find Victim”, “Monitor Health”, “Transit to SAR Operation” and “Track Victim” is modelled in Rhapsody (see Figure 7-7).
In the scenario for the architecture generation we assume that the “Search” Capability shall be replaced by a refined and new “NewSearch” Capability. This change should be applied automatically to the model while the original Capability shall be preserved. Otherwise one could easily just change the Capability itself.

The rules for the change specify that the “MapsToCapability” relation shift for each Operational Activity to the new Capability. This change is of course not limited to that kind of relations or objects. The rules define for each Operational Activity to perform this change without any further conditions, so as to keep this example as simple as possible. Therefore the rules are very similar. As one example see Figure 7-8.

![Figure 7-8: Rule for one local change](image)

This rule contains five different elements. There are class or instance elements representing the objects in the UPDM model. Note that the rules are defined in the same tool/language as the UPDM model to simplify the usage. Due to technical reasons the rules must contain placeholders for the objects of the UPDM model to avoid interference with the UPDM model. To specify for which object a placeholder shall stand for, a “Represents”-Tag is added to the placeholder (see Figure 7-9). If this Tag is empty any object is represented. The blue and green connectors (Rhapsody Dependencies/Flows) indicate a removal (blue) and the creation (green) of an element. In this case a “MapsToCapability” relation is removed and a new one is created between the Operational Activity and the “NewSearch” Capability. The same could be applied also to UPDM objects.

On the right hand side of the screen shot there is the “DrawingToolbar” visible which is customized for Object or Class diagram associated with the Rule stereotype. The elements included in that tool bar allow to drag’n’drop the rule elements (placeholders and relations) into the rule diagram. After adding the elements to the model the “Represents”-Tag for the blocks and the “assoc_type”-Tag for the relations can be specified.
The rules describe up to now only the local changes to be applied to the UPDM model. If a certain final state which could be defined as a pattern which matches several solutions should be applicable to the generated architectures, a “final” rule can be specified. The generation process stops whenever a final architecture is reached or no rules can be applied to that architecture. The generated architecture may depend on the order and number of rule applications and therefore several different architectures could be final. The concept of a final rule helps to control the generation process. To define the final rule (see Figure 7-10) in our example we use an un-typed embargo placeholder which restricts the rule to architecture without any "MapsToRelation" to the original “Search” Capability.

**Figure 7-9:** Represents Tag in a Placeholder

**Figure 7-10:** Final Rule
In this case it is obvious that by applying the rules always the same architecture will result but this is not the general case. Furthermore it is not always the case that the set of rules can generate a final architecture. The first implementation will therefore check if from the current UPDM model a final state is reachable by the defined set of rules. In later versions also the feedback to the UPDM model of the generated architectures as well as the definition of invariant rules will follow. Invariant rules are highly recommended because not only the reachability of final architectures can be detected but also unwanted architectures during the generation process.
8 Architecture Patterns

8.1 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 can 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. Poorly defined legacy systems may need to be retired and replaced in a carefully orchestrated manner along with a requirement to produce an architecture that can migrate from the start to the end state. Therefore, the manner in which the SoS architecture and that of its constituent systems is evolved requires a reproducible and verifiable process. Architecture patterns are seen as one possible way forward in tackling the complexity associated with 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 8-1. 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.
A pattern repository is established (these patterns may be 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. In order to perform analysis of alternative architectural solutions performance data (stored in conjunction with the pattern) can be applied to numerous optimization techniques such as Concise Modeling technology. Figure 8-2 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.

**Figure 8-1:** Illustration showing applicability of architecture patterns over the evolution of a SoS

**Figure 8-2:** SoS Architecture Design Space Explorations
8.2 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 still very much in its infancy. 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. 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).

A pattern repository is required to store and make accessible to the architect, patterns that will help solve problems encountered when modifying the SoS. These patterns may be 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. By establishing a library or repository of architectural patterns for the SoS component systems it is possible to capture current state. More importantly, if additional data such as performance information is stored as part of the architectural pattern it makes it easier to model and simulate future states of the SoS with greater confidence. As well as building a pattern repository it is important to maintain an architecture repository (comprising a proven set of interacting patterns) that represents earlier versions of the SoS. This makes it easier to see how the SoS has evolved over a longer period of time. Once a library of architectural patterns has been created it is possible to begin to explore the impact of alternative architectural solutions.

8.2.1 SoS Architecture Pattern Repository

SoS architecture patterns can be stored within a searchable online database that is indexed according to key pattern attributes. Alternatively, and probably highly desirable, patterns can be stored within a repository that is part of the modelling tool such as IBM Rhapsody. This makes substitution of alternative patterns much easier (however, the current generation modelling tools may not have an extensive search capability) as shown in Figures 8-3 and 8-4.
**Figure 8-3**: SoS Architecture Patterns incorporation within the modeling framework.

**Figure 8-4**: Six stage process showing architecture pattern process

By substituting different architecture alternatives within a Rhapsody model it is possible to explore deeper constituent system interactions than with other architecture representation approaches.
Figure 8-5: Exploring Alternative SoS Architectures

Figure 8-5 shows a number of command and control pattern alternatives which could potentially provide the architectural solution to a part of the SoS or constituent systems. The architect will implement the pattern within the Systems View package in Rhapsody's profile of the UPDM architecture framework. Through a series of UPDM views (Figure 8-6), a SoS's operations can be mapped out to the systems which perform them, and additionally, define and map the required functions to the respective systems. System View 1 (SV-1) shows system connectivity and where architecture patterns are to be captured. However, it must be noted that as one view is altered there may be a knock-on effect to other views and models, simply because higher level architecture patterns do not show the internal interactions of the lower level constituent systems.

Figure 8-6: UPDM Views for SoS Description
8.3 Demonstration scenario

The process of using SoS architecture patterns can be demonstrated by means of a simple test case for a centralised versus decentralised command and control Centre (CCC) architecture. The CCC structure forms the mainstay and defines the success rate in numerous SoS operations including: emergency response, military activities, water management, and ground and air transport systems to name a few. Therefore, the adoption of the correct CCC architecture pattern is critical in the effectiveness of operations, there may be a number of suitable patterns within the repository and it is up to the architect to select the most fitting pattern with respect to the global/local goals and constraints. The global requirements of an emergency response CCC pattern may be speedy response times, robust communication flow, timely decisions at key nodes in the chain of command and so forth. Trade offs can be made from simply looking at the information provided in our pattern templates. A simple example would be looking at the structure of the constituent systems configurations to seek for system redundancy clues or to see whether communications could be costly if there are many interfaces between constituent parts.

This particular use case is abstracted from the CAE, which is focusing on the connections between the SoS comprising CCC, Information Sources, Fire, Police and Medical response resources. The main purpose of this use case is to show the application of SoS architecture patterns and their integration with concise modelling. To apply centralised command pattern to the CCC, user selects <<CentralisedCommand>> stereotype from patterns profile and apply it into the CCC element. Then the CCC element got constraints corresponding to the applied pattern, i.e. the multiplicity value of CCC is restricted to 1. Through the mapping process, the pattern directly affects the elements' connection in technical layer. By comparing the applications of centralised command and decentralised command pattern, it shows patterns alternatives and their effects on the resulting architecture.
9 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_
DM  Design Management application of Jazz. Used to define modeling domains and provide visualization over the web of corresponding modeling data.

DTK  Design management ToolKit. Used for developing new ontology meta-models (domains) in the DM

Elements  Nodes constituting the model data of a project. The model also consists of relations between these elements.

Enrichment  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.

FMI  Functional Mockup Interface

FMU  Functional Mockup Unit

GCSL  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)

GIS  Geographic Information System

GXL  Graph eXchange Language - an xml-scheme which is used by GROOVE.

JIA  Jazz Integration Architecture lays out the architecture for integrating services and application within the Jazz framework.

JTS  Jazz Team Server is the core services provider of the Jazz platform

HTTP  Hypertext Transfer Protocol is the communication protocol over the Internet which is used to connect Web clients (browsers and applications) and servers.

HRC  Heterogeneous Rich Components
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.
<table>
<thead>
<tr>
<th><strong>Project</strong></th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project Publishing</strong></td>
<td>Exporting project-related data stored in a certain tool into the TOOLNET. This mechanism also includes an enrichment function.</td>
</tr>
<tr>
<td><strong>Resource</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Resource Description Framework</strong></td>
<td>It’s a family of W3C specifications for conceptual description or modelling of information that is implemented in web resources. (See: <a href="http://www.w3.org/TR/rdf-primer/">http://www.w3.org/TR/rdf-primer/</a>)</td>
</tr>
<tr>
<td><strong>RDF</strong></td>
<td>See: Resource Description Framework</td>
</tr>
<tr>
<td><strong>RTP</strong></td>
<td>Reference Technology Platform (of CESAR)</td>
</tr>
<tr>
<td><strong>SDK</strong></td>
<td>Software Development Kit. Related to the development environment of services over Jazz/DM.</td>
</tr>
<tr>
<td><strong>Semantic Mediation</strong></td>
<td>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).</td>
</tr>
<tr>
<td><strong>SM Container</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>SMC</strong></td>
<td>Semantic Mediation Container</td>
</tr>
<tr>
<td><strong>SPARQL</strong></td>
<td>SPARQL (SPARQL Protocol And RDF Query Language) is a query language for RDF. <a href="http://www.w3.org/TR/rdf-sparql-query/">http://www.w3.org/TR/rdf-sparql-query/</a></td>
</tr>
<tr>
<td><strong>SysML</strong></td>
<td>Systems Markup Language</td>
</tr>
<tr>
<td><strong>Tool</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
**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
## 10 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>DOI</td>
<td>Reference</td>
</tr>
<tr>
<td>-----</td>
<td>-----------</td>
</tr>
<tr>
<td>[22]</td>
<td>GROOVE</td>
</tr>
<tr>
<td>[26]</td>
<td>LTL</td>
</tr>
<tr>
<td>[27]</td>
<td>SMC</td>
</tr>
<tr>
<td>[28]</td>
<td>DM</td>
</tr>
<tr>
<td>[31]</td>
<td>D8.2.3</td>
</tr>
<tr>
<td>[34]</td>
<td>Dymola</td>
</tr>
<tr>
<td>[38]</td>
<td>CAE</td>
</tr>
<tr>
<td>[40]</td>
<td>D6.5.1</td>
</tr>
</tbody>
</table>
| [41] | CPLEX | CPLEX: “High-performance mathematical programming solver for linear
<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>2013-05-05</td>
<td>50 of 53</td>
</tr>
</tbody>
</table>

11 Appendix A – Architecture Patterns

11.1 SoS Architecture Pattern Repository

There will three repositories for SoS architecture patterns including patterns expressed as documents, IBM Rhapsody models and ontologies. Therefore, this provides patterns informal, semi-formal, and formal representations.

11.2 Document

The document based form is considered as a guidebook for patterns including full pattern information within represented within a designed template. It is important to note that this is an informal representation which can be stored as Word Document, PDF, etc. Figure 11-1 shows Decentralised Command pattern in template.

<table>
<thead>
<tr>
<th>Pattern/Name &amp; Classification</th>
<th>Decentralised Command Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Command System able to take action independently of other commands, and does not rely on communication for achieving mission.</td>
</tr>
<tr>
<td>Capabilities</td>
<td>Command System is able to take action independently of other commands, and does not rely on communication for achieving mission.</td>
</tr>
<tr>
<td>Motivations</td>
<td>Command System is able to take action independently of other commands, and does not rely on communication for achieving mission.</td>
</tr>
<tr>
<td>Limitations</td>
<td>Command System is able to take action independently of other commands, and does not rely on communication for achieving mission.</td>
</tr>
<tr>
<td>Applicability</td>
<td>Level 1 Architecture Pattern</td>
</tr>
<tr>
<td>Structure</td>
<td>(Diagram)</td>
</tr>
</tbody>
</table>

![Figure 11-1: Decentralised Command Pattern in Template]

11.3 Rhapsody Models

The SoS architecture pattern's semi-formal representation is based on graphical notations such as UML, SysML or UPDM models. According to the models, pattern's elements, properties and relations are...
abstracted and formally delivered. Moreover, stereotypes and tags are employed to support patterns’ organisation and configuration. In order to integrate these into IBM Rhapsody, the patterns library are saved as profile (*.sbs format) which allow it to be imported into any Rhapsody project. Figure 11-2 is an example of level-based patterns profile structure showing patterns in different levels with stereotypes, including Intelligence-Command-Response (ICR) pattern, Centralised Command pattern, Decentralised Command pattern and Cascading Decentralised Chain of Command pattern.

![Patterns Profile in Rhapsody](image)

**Figure 11-2:** Example of Patterns Profile in Rhapsody

### 11.4 Patterns Ontology

The ontology-based SoS patterns repository is the formal representation which also include notations using semantic web technologies such as RDF and OWL. Because it represents all the patterns’ information formally, the pattern’s ontology can be used to be the database for an online library which assist pattern’s understanding and selection by providing a serious functions, i.e. searching (keyword searching and advanced searching), pattern’s representation, tree structured navigation, etc. Moreover, through semantic mediation, patterns ontology support pattern’s import and export on IBM Rhapsody. Figure 11-3 is an overall process of patterns repositories integration representing their connection with online library and IBM Rhapsody.
Figure 11-3: Patterns Repositories’ Integration