Conceptual and architecture principles of SoS design and semantic interoperability of systems platform and SoS design Tool-Net

D_8.1.1+D_8.2.1

<table>
<thead>
<tr>
<th>Deliverable Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature</td>
<td>R</td>
</tr>
<tr>
<td>Project</td>
<td>DANSE</td>
</tr>
<tr>
<td>Deliverable ID</td>
<td>D_8.1.1,D_8.2.1</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Contract Number</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>2012-04-30</td>
</tr>
<tr>
<td>Version</td>
<td>1.1</td>
</tr>
<tr>
<td>Contact Person</td>
<td>Uri Shani</td>
</tr>
<tr>
<td>Organisation</td>
<td>IBM</td>
</tr>
<tr>
<td>Phone</td>
<td>+972-4-8296282</td>
</tr>
<tr>
<td>E-Mail</td>
<td><a href="mailto:shani@il.ibm.com">shani@il.ibm.com</a></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>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>Christoph Etzien</td>
<td>OFFIS</td>
<td><a href="mailto:christoph.etzien@offis.de">christoph.etzien@offis.de</a></td>
</tr>
<tr>
<td>Roy Kalawsky</td>
<td>Loughborough University</td>
<td><a href="mailto:R.S.Kalawsky@lboro.ac.uk">R.S.Kalawsky@lboro.ac.uk</a></td>
</tr>
<tr>
<td>Lebeaupin Yann</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>0.00</td>
<td>2012-01-09</td>
<td>Initial layout and TOC</td>
<td>All</td>
</tr>
<tr>
<td>0.1</td>
<td>2012-02-27</td>
<td>Document skeleton</td>
<td>All</td>
</tr>
<tr>
<td>0.2</td>
<td>2012-03-12</td>
<td>Abbreviations and references</td>
<td>14-18</td>
</tr>
<tr>
<td>0.3</td>
<td>2012-03-29</td>
<td>Input from Leonardo and Christoph</td>
<td>5.2, 2.6.1, 2.6.2</td>
</tr>
<tr>
<td>0.4</td>
<td>2012-04-04</td>
<td>Input for SoS Architecture background</td>
<td>2.3</td>
</tr>
<tr>
<td>0.5</td>
<td>2012-04-10</td>
<td>SPRINT background material</td>
<td>2.6.1, 2.6.2, 2.6.3</td>
</tr>
<tr>
<td>0.6</td>
<td>2012-04-22</td>
<td>Updates and new sections</td>
<td>2.4.1, 4, 5, 6</td>
</tr>
<tr>
<td>0.7</td>
<td>2012-04-23</td>
<td>Updates in prep for 1st review</td>
<td>5 and 6</td>
</tr>
<tr>
<td>1.0</td>
<td>2012-04-29</td>
<td>Final version</td>
<td>2.5, 7, 8</td>
</tr>
<tr>
<td>1.1</td>
<td>2012-04-30</td>
<td>SODIUS contrib.</td>
<td>6.7</td>
</tr>
</tbody>
</table>
# CONTENT

## 1 INTRODUCTION

1.1 Overview, Purpose and Scope .................................................................................. 6

## 2 BACKGROUND

2.1 SoS METHODOLOGY CONCEPTS – FROM WP4 ..................................................... 7
2.2 SoS ARCHITECTURE APPROACHES CONCEPTS – FROM WP5 ............................ 8
   2.2.1 What are patterns? ......................................................................................... 8
   2.2.2 SoS Hierarchy of Patterns ............................................................................ 10
   2.2.3 SoS Design Tool-NET Pattern Repository: Describing SoS Architectural Patterns: Template ................................................................................................................. 12
2.3 SoS LANGUAGES CONCEPTS – FROM WP6 ....................................................... 12
   2.3.1 Implications on the tool net ......................................................................... 13
2.4 SoS ANALYSIS CONCEPTS – FROM WP7 ............................................................ 14
   2.4.1 Simulation .................................................................................................... 14
   2.4.2 Statistical model checking .............................................................................. 14
2.5 LESSONS LEARNED FROM OTHER EU PROJECTS ............................................. 14
   2.5.1 SPEEDS........................................................................................................... 15
   2.5.2 CESAR ............................................................................................................. 16
   2.5.3 SPRINT ........................................................................................................... 18

## 3 SoS DESIGN ASPECTS

3.1 INTEGRATION OF LEGACY SYSTEMS ................................................................... 21
3.2 DESIGN FOR EVOLUTION ....................................................................................... 21
3.3 EXTENSIBILITY AND ENVIRONMENT LONGEVITY (LONG TERM SUPPORT) ....... 22

## 4 TOOLS INTEROPERABILITY CONCEPTS

4.1 FOUNDATION STANDARDS OF THE TOOL-NET .................................................. 23
   4.1.1 Support traceability of users’ requirements .................................................... 24
4.2 REQUIREMENTS TOWARD THE PARTICIPATING TOOLS .................................... 25
4.3 INTEROPERABILITY CONCEPTS .............................................................................. 26
4.4 EXPECTATIONS OF TOOLS .................................................................................... 28

## 5 DESIGN MODELLING SEMANTIC INTEGRATION CONCEPTS

5.1 THE “COMMON SEMANTIC” CONCEPT .................................................................... 30
5.2 APPROACHES TO MEET A COMMON SEMANTIC AMONG DIVERSE TOOLS ........ 30
   5.2.1 Semantic mediation ....................................................................................... 30
   5.2.2 Typical Tool-Specific tools-net Flows .............................................................. 32
5.3 MODELLING AND TOOLS DOMAINS RELEVANT FOR DANSE ......................... 34

## 6 ARCHITECTURE PRINCIPLES OF THE TOOL-NET

6.1 THE CONCEPT OF IP PROTECTION ....................................................................... 35
6.2 ARCHITECTURE PATTERNS OF THE TOOLS-NET .............................................. 36
   6.2.1 Components .................................................................................................. 36
   6.2.2 Tools and applications .................................................................................. 38
6.3 THE PLATFORM .................................................................................................... 38
   6.3.1 Agents .......................................................................................................... 39
   6.3.2 Internal Applications .................................................................................... 39
6.4 DISTRIBUTION CONCEPTS .................................................................................. 40
6.5 EXTENSIBILITY OF THE PLATFORM ..................................................................... 42
   6.5.1 Plug-In Components .................................................................................... 43
   6.5.2 Plug-in for Extending the Web GUI ............................................................... 43
   6.5.3 Repository Management .............................................................................. 45
6.6 SUPPORT FOR MULTIPLE META-MODELS ......................................................... 46

<table>
<thead>
<tr>
<th>Version</th>
<th>Status</th>
<th>Date</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Final</td>
<td>2012-06-04</td>
<td>3</td>
</tr>
</tbody>
</table>
Content of Figures

Figure 1: The DANSE systems engineering lifecycle ................................................................. 7
Figure 2: SoS Hierarchy of Patterns ...................................................................................... 10
Figure 3: Example Architecture Pattern (reproduced from [17]) ........................................... 12
Figure 4: SPEEDS engineering environment ..................................................................... 15
Figure 5: RTP - Overview .................................................................................................... 16
Figure 6: RTP - Common View ........................................................................................... 17
Figure 7: SPRINT SII and SSI architecture, and the use of physical devices (PD) .......... 18
Figure 8: Semantic mediation concepts in SPRINT ............................................................. 19
Figure 9: Distributed SII ....................................................................................................... 20
Figure 10: Distributed simulation and co-simulation management over the internet ecosystem ......................................................................................................................... 20
Figure 11: Point to point vs. "hub and spoke" ..................................................................... 26
Figure 12: models, relations, ontologies and rules................................................................. 27
Figure 13: OSLC servers and clients .................................................................................... 27
Figure 14: Exchanging models from different tools via the common semantic model. ........ 31
Figure 15: The flow of model data into tools-net server from Rhapsody .............................. 31
Figure 16: Mediation and links ............................................................................................ 32
Figure 17: Tool-specific RESTful protocol between tools-net server and tools ................... 33
Figure 18: Servers interoperability over the internet (as done in SPRINT) ......................... 35
Figure 19: Architecture patterns of the tool-net ................................................................. 36
Figure 20: Jazz components ................................................................................................. 39
Figure 21: Tools interoperability through the DANSE tools-net servers ......................... 40
Figure 22: Specific configuration of a distributed collaboration among two partners over the internet ......................................................................................................................... 41
Figure 23: A generalized collaboration scenario depicting the management of multiple distinct projects ................................................................. 41
Figure 24: GUI view of a distributed modeling data over corporates and tools .................... 42
Figure 25 – Distributed DANSE tools-net server sequence diagram ................................. 45
Figure 26: Authoring ontology with the DTK ...................................................................... 47
Figure 27: MDWorkbench interfacing models for the tools-net ............................................ 48
Figure 28: WP8 workplan .................................................................................................... 49

Content of Tables

Table 1: Traceability of users’ requirements to basic and implied technologies ....................... 24
Table 2: Tools proposition by IAI, not discussed yet. ............................................................. 28
Table 3: Generic Adapter Component Feature List .................................................................. 42
1 Introduction

1.1 Overview, Purpose and Scope

This document is a combined delivery of D8.1.1 and D8.2.1 for the WP 8 ("SoS Tool Net and DANSE exploitation") tasks T8.1 ("Formalize the main concepts part A") and T8.3 ("Create and validate architectural decisions"). In this document we make an effort to define the guiding principles of the supporting tool-net platform on which to carry out SoS design within the DANSE eco-system.

Clearly, the document depends on other DANSE deliverable, specifically the requirements document D3.1 [31], and the CAE. The fact is that CAE document is due by M12 and D3.1 is due by M6 so the dependency on these efforts is problematic.

The document combines a discussion of the concepts and of the architecture to support these concepts. The high level of coupling between these two lead to combine them into a single document and to follow up new revisions of this document at each of the future deliveries planned for either concepts or architectures.

The concepts of SoS design are a culmination of concepts following each of the major technological challenges of DANSE. That is the background for DANSE setting up the conceptual context of the Tool-Net. Each of these challenges is managed by one of DANSE technological work-packages (WP-s), which will also provide these concepts.

DANSE follows up on past and present projects whose relevance will be outlined and lessons learned from these will be also described as part of that background – specifically the SPEEDS [5], CESAR [6][7] and SPRINT [7] EU projects.

The collaborative context in which DANSE’s Tool-Net operates is also rich with standards that DANSE adopts. Moreover, the platform of choice that respects these standards and which will be used to implement the collaborative DANSE platform of tools is the open Jazz [1] platform by IBM. All of these will also be discussed in the background chapter.

The next chapter discusses the SoS design aspects, which would usually start with legacy constituent systems, support evolution of that design into new architectures and new constituents. The SoS lifecycle covers these first two features of the SoS design and the need to support a very long lifecycle which requires first to handle the continuity of the support for the lifecycle over long periods as well as ensure to the constituent developers that indeed the support will be accessible and usable for the entire longevity of the system.

We next deal with tools interoperability within the DANSE tools chain ecosystem. This chapter is technical in the sense that interoperability among tools and the SoS tool-chain must be well defined and anchored in technologies and standards that will make worthy the investment in time and effort in tools adaptation.

The semantic dependency among tools and how to deal with that in DANSE is dealt in the next chapter, identifying the domains in which semantic integration among tools is required. Here we will build on top of semantic web technology as adopted in SPRINT and which we hope can also be used in DANSE.

The next two chapters deal with the DANSE architecture and the execution plans for developing and deploying the DANSE service platform – a distributed and collaborative ecosystem based on the Jazz open system. We conclude with a summary.
2 Background

As a background to this document we bring up initial ideas from the technological and use-case work-packages (WP) of the project which shed some light on how the tool-net and tool-chain capabilities are needed. These contributions are made by WP leaders.

2.1 SoS methodology concepts – from WP4

The distinctive feature of the DANSE SoS engineering lifecycle is that it relies upon an evolutionary process. (See Figure 1) Classical models of systems engineering, such as the V Model rely upon a top-down view of design and a linear, once-through approach to systems engineering. The SoS of interest to DANSE are an aggregation of pre-existing ‘legacy’ systems, with added functionality, and new constituent systems developed specifically for the new SoS. This has consequences on the systems engineering lifecycle.

![Figure 1: The DANSE systems engineering lifecycle](image)

The classical approach to design aims at producing systems that are "correct the first time", that is, systems that satisfy all constraints and optimize given objectives without resorting to expensive redesigns after a system has been placed in operation. Because of the inherent complexity of an SoS in terms of sheer number of independent agents that are part of the SoS, and because an SoS is usually an opportunistic aggregation of legacy systems, obtaining the correct behavior when first placed in operation is highly improbable. For example, we experienced the many iterations and many months it took to bring up the Heathrow airport (UK) after renovation and the Bay Area Rapid Transit system (USA). We believe that an evolutionary approach to correcting the behavior of an SoS is necessary. The DANSE methodology will shape the behavior of a SoS over time as more is learned about the constituent systems behavior and the emergent behavior due to their interaction, so that the system becomes “correct by evolution.”

Figure 1 shows five system-of-systems engineering processes and a set of constituent systems engineering processes. The first two processes are complementary:

1) Model SoS behavior – Simulation models run continuously during the lifecycle of the SoS and performance indices obtained by simulation are compared as they become available with operational data. Since most SoS are too large and complex to run comprehensive model validation experiments, validation cannot be but an incremental process. Modeling, validation and design corrections based on the proven technology of contract-driven design developed in SPEEDS [5] will be performed continuously.

2) Operate the SoS – An SoS will operate continuously and experience incidents of emergent behavior on a regular basis. When the operating data diverges significantly from the predictions of the operational model, the operational model must be adapted to cope with the inaccuracy that may lead to negative emergent behaviors.
3) The lower three SoS engineering processes embody a learning cycle and support steady performance improvement for modeling and operations over time, as the SoS evolves:
   a) Define potential needs – This task is based on a two-step process:
   b) Identifying a need - Ongoing analysis of system performance can lead to the determination of the needs in two ways: Either: 1) modeling and analysis suggest that better performance is possible or 2) the system's behavior differs significantly from the current prediction obtained by the DANSE operational simulation. Incidents of unexpected emergent behavior may be negative (indicating performance problems or malfunctioning), or positive (indicating opportunities).
   c) Analyzing the cause – Root-cause analysis is supplemented with a knowledge base of previously analyzed incidents to support rapid identification of recurring patterns. The knowledge base becomes increasingly sophisticated as the system and its operators "learn". The validation of a causal mechanism is strongly supported by the DANSE simulation technology.

4) Analyze possible architecture changes – Analyzed needs drive changes in the system to exploit opportunities or correct problems. A change may suggest fundamental changes to the architecture of the SoS or to the architecture of constituent systems. A key architectural tool is the predictive modeling and simulation capability used to compare architectural alternatives. Simulation also has an important role in the verification and validation of proposed architectures since logical validation can offer significant savings in testing.

5) Influence and implement changes – Types of changes include: 1) Updating a simulation model to give more accurate estimates of system behavior, 2) Modification of a contract between the SoS and one or more constituent systems. 3) Re-factoring the architecture at the SoS level.

The Vs in the lower part of the lifecycle diagram represent systems engineering efforts at the level of constituent systems. Constituent systems are typically legacy systems that were developed using a variety of engineering processes. During SoS operation, when the capability engineering process identifies a needed capability enhancement it may influence the various constituent systems to implement a change. Engineering processes at the constituent systems level are not dictated from the SoS level, but must be coordinated with the overall SoS goals applying contract based technology.

Most SoS are initially created as opportunistic aggregations of legacy systems after studying and mapping the potential constituent systems, external actors and other significant features. Typically, most or all of the constituent systems have been operating for some time prior to being inserted as part of the SoS.

In the Initial Phase, a descriptive model of the system is created first, using the DANSE language and modeling approach. The DANSE approach to creating a descriptive model is unique in that it applies “contracts” to represent key performance and interface requirements. Contracts then form the basis for the dynamic, predictive SoS models applied throughout the management, operation and evolution phase.

Given the special nature of SoS described so far, the research done in the DANSE methodology area will integrate all simulation, architecture alternative selection and analysis techniques developed in the project. This will be done using the DANSE language and modeling approach with a focus on the two distinct, overlapping, phases of its life cycle:
   a) Initial phase - mostly dedicated to the design of the “new” SoS.
   b) Phase S - management and continuous evolution, where the design is now a continuous activity, interleaved with the continuous operation and evolution.

2.2 SoS architecture approaches concepts – from WP5

Patterns are the foundation of architecture approaches for SoS in DANSE. Emerging from the original work of Alexander [32] into the state of the art in programming and systems architecture [14] we adopt and adapt this approach in the extended problem domain of SoS in DANSE.

2.2.1 What are patterns?

The challenge for the systems of systems (SoS) architect is how to build a model of the complete SoS in way that allows them to describe, model and analyse the system. This is where the field of ‘patterns’ has an
Patterns are short structural descriptions (and models) that capture solutions to previous problems practices that have been successful. It is important to note that they are not complete software modules but a form of blueprint that can be applied to specific problems. Patterns frequently capture best practice solutions to problems but they may not represent a complete solution. However, patterns typically provide the starting for the systems architect or systems designer. The use of patterns in software design is a well-practiced field but their use outside detailed software design is less well understood. The use of patterns can significantly help to explain or understand a complex concept or interaction by separating the key concerns away from a particular implementation. This is, in many ways like the platform independent model referred to in model-based systems engineering. Consideration of a system or SoS at a higher abstraction level can help the systems architect or design get a fuller picture of the system(s) of interest. In some systems it is only the interfaces that are exposed to other systems (the detailed lower level system specific functionality is embedded in the system) which makes it more important to understand the system at an abstract level. It is all too easy to get 'bogged' down in the minutia of a particular system and loose sight of the bigger picture. This is the opposite of the old adage 'I can’t see the wood for the trees’ – in this respect it is important to ‘I want to see the trees and not the wood’.

Therefore, we can summarise that patterns help the systems architect and designer to focus on the higher level (somewhat abstract) interactions between the system(s) and the(ir) environment. We can consider a system (or SoS) almost as a continuum from the macroscopic SoS level through constituent systems all the way down to interactions at the molecular level. This might seem to be an absurd scale but we are dealing with interactions that can and do reveal themselves at different levels of the system representation framework. In the software community the notion of ‘design patterns’ is reasonably mature where common design patterns have been defined into creational, structural and behavioural categories. Other design patterns have emerged that assist in user interface design [15]. These patterns are typically very specific and enable the software designer to rapidly create better object-orientated software at the detailed working of a specific system component.

Patterns can be considered at higher levels where instead of being confined to being within a single system boundary they can be applied across system boundaries so they help the focus of attention at the higher system-to-system level. The ability to do this is very important when in the context of SoS it is necessary to consider a large scale enterprise in the form of a SoS where the SoS evolves in ways that was not originally conceived or where parts of the SoS are actually services rather than system components. At the higher SoS level we are particularly interested in how the constituent systems are coupled together where it’s not just the nature of the physical connections that matter but the quality of service becomes a factor. Of particular interest is the point that the SoS will constantly evolve – in terms of the interface coupling between the system it is essential that changes to the interface can be accommodated whatever they are. If systems can be designed this way they are less likely to be restricted in the future. Within DANSE the dynamics of a SoS is of particular interest – patterns may be a good way to capture and represent the evolutionary (dynamics) aspects.

A pattern can be expressed using both human language such as text, and more formal representations such as Unified Modeling Language diagrams (UML). In describing the use of a specific pattern it is also possible to provide code fragments to illustrate a design solution. However, it is not the remit of a pattern to provide a fully coded example.
2.2.2 SoS Hierarchy of Patterns

When considering a model for a SoS it is sometimes convenient to think in terms of a three-layered stack comprising operational, systems and component models respectively, refer to Figure 1. At the highest abstract level we have the operational model that defines the overall system architecture – which is completely independent of the way the underpinning systems and services are implemented. In essence this is the architectural framework on which the SoS exists. At the next layer down are the underpinning systems models – these are also implementation independent but enable a more specific model to be constructed comprising individual system models. At the lower level we have the component models that are implementation specific and encapsulate all the variables of a particular solution.

![SoS Hierarchy of Patterns](image)

Figure 2: SoS Hierarchy of Patterns

Figure 2 shows that there is some overlap in the boundaries between the three layers. In terms of patterns it is convenient to think of three pattern categories: Architectural, Design and Interaction. At the architectural level the patterns are very much concerned with the top most level blueprints (expressed in an architectural framework such as DODAF, MODAF or NAF), whereas at the design pattern level the designer is able to represent system design at the abstract level (without considering specific interface standards). The Design patterns can also be used at the more traditional level of defining the enabling software in object-orientated structures (and use UML and SysML). At the lowest level design patterns are most associated with interaction design and coupling. These patterns will describe patterns for SOA, contracts and other interfaces between systems.

Arguably, one of the most critical elements of a SoS is the interaction between the constituent parts – at a discrete messaging level or through the coupling between system interfaces. Examples of system coupling can include a) tightly coupled interfaces such as service-orientated architectures, contract based interfaces and b) loosely coupled interfaces that operate but do not have an external controlling influence.

Analysis of most system architectures will reveal many patterns. When applied correctly patterns play a huge part in architecting complex systems, they can be used to express common elements in a system design in a way that makes implementation easier. Also well-constructed patterns can be re-used in other places and make it easier to document and maintain existing systems through the use of a library or catalog of patterns.
Patterns are parameterised collaborations; in other words, they are a group of collaborating objects/classes that can be abstracted from a general set of modeling scenarios. Patterns are an excellent means of achieving re-use and building in robustness. As patterns are discovered in any new project, the basic pattern template from previous engagements can be re-used with the appropriate variable names modified for the current project. Patterns generally describe how to solve an abstract problem, and it is the task of the pattern user to modify the pattern elements to meet the demands of the current engagement. Before using a pattern it must first be created as a standard UML diagram and then saved as an XML pattern file. This XML file can then be imported as a UML Resource that can be used in any model. A pattern describes a problem, which occurs over and over again and then describes the core solution to that problem. In the context of SoS patterns can be defined in terms of interfaces and objects. A typical pattern has four key elements: Pattern name, Problem being solved, Solution to the problem and the consequences.

The SoS Design Tool-Net will need to make provision for the storage and retrieval of various patterns from a repository (or factory). These patterns will enable reuse of important elements used in the architecting of SoS. Research undertaken in the DANSE project will look to extract or mine reusable patterns from the industrial test cases and the concept alignment example and represent these in the pattern repository. Other examples of good patterns will be mined from other projects and made available through the repository along with guidelines for their use.

The SoS architecture description is likely to be very complex, because it comprises a large set of components and relationships between them. Also, based on stakeholders' concerns, different parts of the SoS architecture will be relevant for different stakeholders. This requirement will necessitate the SoS Design Tool-Net to represent information at an appropriate level of detail to allow analysis of the SoS to be undertaken. One of the important applications of patterns is the abstract description of the logic for static analysis of the SoS architecture to support impact of change analysis. However, for dynamic analysis techniques patterns will need to be developed (or mined) to cover approaches such as process algebra and data flow networks in order that sub-optimal parts regarding the logics of architecture can be identified with these techniques.

An example of a reusable architectural pattern is given in Figure 3. This has been mined from several examples [16]. It serves to show the generality of using a pattern to describe a set of operations that pertain to a number of similar high-level tasks.
2.2.3 SoS Design Tool-NET Pattern Repository: Describing SoS Architectural Patterns: Template

The SoS Design Tool-NET will need to include the following information for all patterns in the repository. Imposing a strict structure on the repository will make it easier to catalog the patterns as they are mined and made available for re-use. The following information is required to define a pattern and make it useful to other users:

- Pattern name & classification
- Intent
- Also known as
- Motivation (Goals, Capability, Limitations)
- Applicability
- Structure
- Participants
- Collaborators
- Consequences
- Implementation
- Sample Code
- Known Uses
- Related Patterns
Conceptual and architecture principles of SoS design and semantic interoperability of systems platform and the SoS design Tool-Net

Concepts of the SoS languages which are developed in WP6 focus in this phase of the project to dynamicity aspects and probabilistic specification.

The dynamicity relates to the overall SoS lifecycle and explicitly contains the integration of new constituent systems to the SoS, the change of such systems and their removal. This aspect is covered by the concept of the evolution of a SoS. Due to the fact that the participating constituent systems are independent in several dimensions, their change cannot be specified like in the classical system engineering domain. The reconfiguration of the SoS itself is therefore dynamic and could in general lead to a unbounded number of states for a reasonable time frame. Analyses of this evolution are therefore based on probabilistic assumptions/measurements for the SoS. Also the ability to handle future situations is based on such probabilistic methods.

Constituent systems collaborate in order to achieve the (global) goals of the SoS but also follow their own (local) goals which might be competitive with some of the global and/or other local goals. These systems participate in the SoS only because they gain benefit out of this collaboration. To optimize their local goals these systems interact with the others depending on the current operational situations. If the situation changes all interaction relations may also change.

A subset of such interactions of constituent systems result in expected or even unexpected emergent properties or behaviour. Such emergence is not generally positive or negative but could support a set of goals or not. The estimation of possible negative or positive effects is also one analysis target but even the current modelling formalisms lack in representing such properties or behaviours. Therefore the modelling concept is needed first before an analysis method could support the identification of such properties. The modelling of such effects is in the focus of later investigations on this topic and will result in sound modelling and analysis capabilities.

The current SoS modelling concepts are based on classical system modelling languages for the constituent systems like UML and on several approaches from the state of the art for the interaction and evolution of dynamically interacting systems in an operational (short term) time scale. The current candidate of modelling the long term view is the graph transition modelling formalism which seems to be applicable to cover all relevant evolutionary changes.

Furthermore the SoSs DANSE focus on are not new in the sense that they are constructed from scratch but they are already existing. The challenge is to support the introduction of new systems to an existing SoS or to upgrade a already integrated constituent system.

### 2.3.1 Implications on the tool net

The continues nature of the SoS development process implies several different classes of roles for tools connected to the tool net. One class covers the different development tools which include modelling as well as requirement tools. A second class contains different kinds of analysis capabilities which should be provided via the tool net. Those analysis services have to comply with the modelling language and methodology. Simulation tools have also to be integrated in a similar way via the tool net. Furthermore, the availability of data in some kind of repository (centralized or decentralized) is mandatory. Last but not least a role of tools which monitor the SoS during the continues development is required.

Since the methodology has to take different views, e.g. related to different abstraction levels of the SoS, into account, the tool net must support to manage those views to all data and services provided. Data and service linkage based on unique identifiers is a minimal requirement for the interoperability. WP6 will develop a sound modelling language supporting the different modelling, analysis, simulation and observation processes. This language builds one of the building blocks for the semantic interoperability for exchanged data as well as the methodology defines the semantic of services using this data as input or output.
2.4 SoS analysis concepts – from WP7

2.4.1 Simulation

From the tool net perspective, we identify the following relevant concept regarding simulation:

- Support for distributed simulation;
- Connection with off-line data sources;
- Connection with on-line data monitoring.

We discuss now the above concepts with more details.

Support for distributed simulation across different simulation tools come in three forms:

- Support for time synchronization;
- Support for connectivity and data exchange (communications) between components located in different tools;
- Support for agent migration between tools in agent-based simulations.

The tool net infrastructure should support the kind of interactions between tools described above, although the specific protocol to guarantee time synchronization will have to be implemented by the specific tools.

Connection with shared data sources is a critical aspect for SoS design, analysis and operations, in order to avoid producing erroneous and misleading information. One of the overarching technologies in this respect is represented by the Geographic Information System (GIS). GIS technology combines mapping software with database management tools to collect, organize, and share many types of information. Data is stored as thematic layers in geo-databases (data identified by its location coordinates) that can be accessed and shared from the field, within a department, and across an entire enterprise. The user decides which layers are relevant. Utilities typically combine utility layers with land base, parcel, street, land-use, and administrative area layers. GIS technology can be used to

- Track and report on assets in the network inventory;
- Generate inputs into modeling software;
- Create a common operational picture for access to network operations information.

Esri [29] provides an online cloud-based collaborative content management system called ArcGIS Online. The integration of GIS technology with the simulation is an example of off-line data source integration. Other examples are the weather forecast information services and aircraft situation display to industry (ASDI), for air traffic management simulation and control decision support.

On the other hand, integration of on-line data monitoring services are necessary for control decision support for city traffic management, where constant monitoring of crossroad situation is a key aspect to keep simulation analysis accurate.

2.4.2 Statistical model checking

Statistical model checking requires the integration of simulation tools and statistical hypothesis testing frameworks. This integration can be effectively achieved through an interface API between the tools.

2.5 Lessons learned from other EU projects

Some EU projects that have completed or are still going on have relevant lessons learnt for DANSE, with respect to the tools, interoperability, modelling approaches, data integration and so on.

2.5.1 SPEEDS

The SPEEDS engineering environment combines several modelling tools together with various analysis tools and newly created modelling tools. The engineering activities are guided by a Process Advisor. All are working on a common data basis the SPEEDS Repository (see Figure 4).

The SPEEDS bus is responsible to connect all kind of tools and services to be able to work together. For this integration of services the SPEEDS Bus offers a well-defined API (the SPEEDS Bus Service Layer) and all tools are connected to the SPEEDS engineering environment through this API. Using this API the SPEEDS Engineering Environment is open to integrate additional tools – design tools as well as analysis tools.

While the SPEEDS bus is the central point of tool interaction the SPEEDS model repository is the only data storage for all tools, the storage itself being a tool providing its service.

Each tool stores its data in the repository. This may be the system’s HRC model processed in common by all tools, a tool proprietary data (using a proprietary tool format) or analysis results. Further more the tool may add links to other related files, e.g. an analysis file or a HRC [19] representation of the tool related model. Hence the SPEEDS Repository stores all kind of data as files and furthermore relation among several kinds of data (for example proprietary tool data linked to HRC representational or refinement relations between components, etc.). The structure of the repository, i.e. the kind of data and the kind of relations among the data, is described in an XML file, which is used by the repository to manage the stored data.

![SPEEDS Engineering Environment](image)

**Figure 4**: SPEEDS engineering environment

A drawback of this approach is that the implementation of the SPEEDS API is a prerequisite for the integration of a tool in the SPEEDS engineering environment, while a more flexible approach where increasing tool interoperability is incrementally achieved based on market needs and strategies is likelier to gain acceptance and success. A disadvantage of the SPEEDS approach is also represented by the common model exchange language (HRC), which requires a significant initial effort to achieve interoperability. By contrast, the model transformation approach advocated by SPRINT allows incremental achievement of the model exchange objective, while reusing existing solutions.

### 2.5.1.1 Lessons Learnt

One of the major lessons learnt in SPEEDS is that a single modelling language to be shared by all tools is not practical. Yet, SPEEDS advocated against the point-to-point tools interoperability, and that concept is also adopted by the DANSE tool-net.

The single language concept in SPEEDS generated an inhibitory cost for tools to interoperate and integrate. The Speeds Engineering Bus is indeed an incarnation of interoperability concept which DANSE will employ. Yet, DANSE has a highly multidisciplinary environment which must support multiple tools domains with possibly vast differences and will need to depart form the single model concept as well as learn that even in the same domain – this is not a good idea. Moreover, DANSE is based on the distribution of resources and cannot tolerate a single repository, yet the logical repository model may server DANSE as well.
HRC has in-fact been adopted as one approach to integration of tools, designs and applications via a common language for defining contracts of components, extending that to the SoS level where components are constituent (possibly complex) systems.

2.5.2 CESAR

The CESAR [6] Reference Technology Platform (Figure 5) is a generic model-based tool integration platform composed of a set of interoperable tools, methods and processes designed to improve the development of safety critical embedded systems.

For exchanging data between tools, several connection types are proposed:

- Point to point data conversion:
  - import / export functions provided by the tools themselves, e.g. SCADE / Simulink
  - model transformations, written in model transformation languages (e.g. ATL), to be executed by model transformation engines

- Common meta-model integration, based on the mapping between tool native models and the RTP Meta-Model.

- Data integration by linking: This is a lightweight data integration mechanism. It allows the creation of links between elements of models and the persistence on these links in a dedicated link repository, without impacting the models themselves. These links can be navigated by using a programmatic API or a GUI viewer.

These different integration schemes are not exclusive in general, they might co-exist. In CESAR a combination of data-linking and providing a common view to this data is realized as reference implementation (see Figure 5).

![Diagram of CESAR Reference Technology Platform](Figure 5: RTP - Overview)
All data created usually by specific development tools are linked over the border of proprietary tools with each other. To handle the semantic differences of these tools, each link is associated with a view to the referenced element based on the common meta model (RTP-Meta-Model). This enables a homogeneous view for browsing and traceability purposes as illustrated in Figure 6. Note that due to the proprietary and distributed nature of the elements in some cases an additional Link Repository is required to store links between elements.

![Figure 6: RTP - Common View](image)

This common understanding of generic concepts allows on the one hand implementing tool (and domain) independent analysis services (as depicted in Figure 5 as green services) and on the other hand creating also tool and domain independent library of development process fragments. This CESAR Process Library allows to instantiate a development process based on methods and development artefacts predefined in the common meta model.

The combination of data-linking and common view does not inhibit to exchange data in its original format but enables the reuse of process and methodology pattern in different domains. A domain/company/process specific view to the linked data is a feature which was not in the scope of CESAR but often requested by different users. The creation of the common view is technically located in the adaptors which connect the tool to the RTP. This is less flexible than the way SPRINT [7] handles the data representation issue but aligns the development data representation directly with the methodological predefined process artefacts.

### 2.5.2.1 Lessons Learnt

A summary of the CESAR lessons learnt is provided by Hein and Ritter [30] and makes much sense with respect to DANSE. For instance, understanding that internet protocols and methods should have been used and adopted by the tools rather than inventing (even via specifications) new APIs. The ideas of adapters shared by tools in same domain, and the linking of elements in the models of different tools as well as the use of meta-modeling are good ideas to adopt. Yet, the semantic-web technologies evolving today seem to be more appropriate than the model transformation methods used in CESAR. A general conclusion reached by CESAR consortium is that radically different interoperability techniques may be needed concurrently, depending on the level of granularity of data to be shared, and on the clustering scheme of tools in the engineering process. For example, CESAR demonstrated the relevance of using model transformation to a common metamodel for interconnecting Simulink and Eclipse/Papyrus UML modelling tools. This is fully complementary to using high level semantic links between the (Simulink+UML) model supported by this cluster of tools, and other design models supported by a different cluster of tools.
2.5.3 SPRINT

SPRINT started October 2010 and marked 3 problem domains in Systems Engineering over the internet:

- Internet of Things (i.e., physical devices)
- Internet of Engineers (i.e., people collaboration)
- Internet of Design elements (i.e., tools interoperability, data integration and a distributed tool net).

Of these three challenges, the first one is totally not relevant to DANSE, so we will concentrate on the other two only.

Lessons learnt from SPEEDS at the time of SPRINT conceptualization taught the SPRINT architects that a single modelling language cannot serve all purposes. Like CESAR, SPRINT also recognizes that it is not practical to make all tools to interoperate with each other and therefore, SPRINT adopts the vague notion of semantic specifications interoperability (SSI) as a core technology to be developed and installed over a distributed collaborative environment consisting of the Semantic Interoperability and Integration (SII) platform.

SSI is based on the notion of “Semantic mediation” and adopts the Semantic Web technologies of RDF [3] representation of models and meta-models, where meta-models are defined in the Web Ontology Language (OWL) [26]. The SII is also based on Semantic Web technologies adopting the RESTful [9] API over the internet as means to communicate among tools and applications, defining internal model elements as web resources, exchanged as RDF model data according to OWL meta models, and adopting the OSLC (Open Services for Lifecycle Collaboration [8]) initiative and an evolving standard specification.

The general SPRINT architecture is depicted in the figure above, where the SSI/SII platform and semantic layers provide the equivalent of the CESAR RTP or the SPRINT model bus. The SII platform in SPRINT – like DANSE – uses the Jazz [1] open platform by IBM. Tools interact with the platform via OSLC protocols, or a variant of that protocol when massive model exchanges are required. With OSLC, tools and applications interoperability is practically solved. The remaining big issue is how to make sense of model data communicated over that OSLC specifications and RESTful API over the internet. This is the role of the semantic mediation functionality of the SSI – which is still in progress at this time.

![Figure 7: SPRINT SII and SSI architecture, and the use of physical devices (PD).](image-url)
Model integration via the semantic mediation is a middle of the way approach between the point-to-point approaches where all tools communicate with all others, to the single common modelling language advocated by SPEEDS. In a way, the semantic mediation answers also the preferred middle of the way approach that CESAR learned from its experience and is shown as the 3rd choice for model integration in Figure 8. The essence of the semantic mediation is to use ontological descriptions of models that can be related to each other in simple as well as in complex ways defined via rules. A model may be related in more than one way with other models, depending on their common domains.

Intermediate common models may also be stored in a repository to serve as references, or just be used for on the fly transformations and thus to never be realized, nor to be required to serve as a single common model like the SPEEDS case.

Another aspect of SPRINT is the distribution of model bus repositories (the SII platforms) over multiple nodes on the internet and the proper management of data isolation to allow collaboration among multiple corporate partners, while maintaining their intellectual property rights. The distribution architecture is depicted in the next Figure 9, while the distributed co-simulation is described in Figure 10.

SPRINT also advocates the use of the standardized W3C Linked Data [27] that has also been advocated by CESAR though via different technology than this W3C standard, which is an integral part of OSLC and which has a direct meaning with respect to semantic mediation – that related modelling resources can be linked to each other, made related via links to some common reference model, and use for proprietary models within different interoperable tools.
2.5.3.1 Lessons to be learnt

Unlike CESAR and SPEEDS, SPRINT is yet to teach us new lessons. The challenge of semantic mediation and its effectiveness with the integration of models from different tools will serve DANSE very well. The adoption of the semantic web technologies by SPRINT matches very much the present trends in the market and the use of Jazz to host that platform seems at present very much in line with DANSE. The challenges in DANSE are orthogonal to SPRINT in the sense that they tackle unique issues of SoS. Yet, the plurality of modelling domains in DANSE will require a powerful and flexible semantic interoperability such as the semantic mediation of SPRINT can serve well.
3 SoS design aspects

DANSE DoW [20] states as follows the major technical innovation points of the project which should advance the state of the art in SoS design:

1. A new evolutionary, adaptive and iterative SoS life-cycle model
2. Development of new formal semantics for SoS modelling
3. A formal method for “correct by evolution” analysis
4. High-level behavioural simulation based on SoS abstraction
5. Methods and tools to allow optimization at the global and local SoS level

The technical challenges of DANSE to meet these innovations are then divided again to 5 topics:

1. Methodology
2. Modelling
3. Simulation and Analysis
4. Architecture
5. Tool net

Each of the first 4 technical challenges affects the tool net through the tools by which it can be designed and implemented. For instance, methodologies of SoS design would be articulated via some modelling languages and have some supporting tools that will participate in the DANSE tool-net. Likewise is the modelling challenge of SoS, integrating existing constituent systems and evolving that to new versions of constituents as well as new constituents altogether – such that were not considered in past incarnations of that SoS. Modelling tools will enable the articulation of modelling concepts of SoS into practical languages whose artefacts will be managed by tools much as the tools managing methodological artefacts. The interplay of methodologies and modelling tools may mean that methodologies will dictate the process of modelling, so that a coupling of the methodologies and modelling artefacts will be executed over the tool-net.

Such interplay should be made explicitly during the project requirements compilation. Yet, it can be viewed at this point already, that in both cases we refer to languages by which methodologies as well as modelling concepts can be expressed and articulated (using tools) to concrete models. That is the common tool-net view over the entire set of DANSE technical challenges.

Similarly we can look at the architecture and the simulation and analysis challenges. Tools may be not only such that manage models according to specific languages, but also what we term applications which may not manage models, yet they may perform some value-added function such as simulation or some analysis. All the same, a tool may do both – manage models and perform value-added functions over these models.

In the next sections we will briefly reflect on the behaviour of the Tool-net to support the related aspects of SoS design.

3.1 Integration of legacy systems

We have evidence in the DANSE use cases and concept validation for the fact that the SoS is not a “green field”, meaning that we start with an existing operational SoS which may be treated by its partners as such, or not. Yet, there are always the constituents that operate in that SoS space and which (if not yet so) will become constituents of the SoS designed using the DANSE solutions.

From the requirements document, we see for the Air Traffic Management (ATM) use case, it is observed that “Heterogeneity of the constituent systems (from sensors to decision making through situation awareness)” as well as the fact that such a system needs to compensate for existing dated constituents as well as with the need to develop new solutions in agile methodologies.

More specific, user requirements for working with legacy systems are made explicit such as user requirement UN-05-001.

The methodology work package has already outlined (see 2.2 “SoS methodology concepts – from WP42.1”) the understanding that an SoS starts with existing constituent systems operating as part of some larger
operational space (kind of a “present” SoS), or such which are only considered to be aggregated into an SoS, possibly as part of an “opportunistic” approach to SoS.

Per the tool-net, legacy constituent systems have their own tool-net supporting their lifecycle from design through deployment to operation and on. Adapting legacy systems and their legacy tools into a different tool-net may be very desired, but not necessarily effective and worthwhile the cost and efforts to do that. In fact, there may be dozens of tools and methods which legacy constituent systems employ and the complete assimilation of the legacy methods into a new methodology may be inhibitory and therefore that should be avoided.

No question is that some part of the legacy constituent system has to be integrated into the SoS development methodology and the tools to handle that part of the system to also integrate into the SoS tool-net. A careful evaluation of the tools should consider whether they are required to be part of the SoS toolnet according to this understanding.

Important question is what part of constituent systems transcends into the SoS design eco-system and thus must be handled through the tool-net tools. That will be answered by the methodology as it evolves. We can safely say that there would be some interfaces through which a constituent system takes part in the SoS, that these interfaces will represent some abstraction of the constituent systems, and that some contracts language will be used to specify these interface so that analysis and optimization tools – for instance – will be able to contribute to the design of that SoS.

3.2 Design for evolution

In the architecture approaches introductory section - as well as in the methodologies introductory section, the design for evolution is a central design concept. This concept is also depicted quite expressively in Figure 1. It articulates the realization that the SoS is not constructed once (perhaps following some lengthy iterative design process within which the system’s concepts evolve) but there is always some part (or the entirety) of the SoS that is evolving and continuously changing. This evolution can cope – for instance - with emergent behaviours, as a results of new insights based on simulations, or also as the SoS is optimized.

The tool-net is therefore never stopping to serve the SoS as it evolves. This means first that the tool-net longevity is significant unlike any other situation know in modern engineering (see also next section). It also means that the tool-net is evolving by itself by continuously accepting new tools and as older tools are discontinued.

3.3 Extensibility and environment longevity (long term support)

An important result of this discussion is that the tool-net must be designed to cope with an evolving and “ever lasting” environment. The essence on which the tool-net will be built should be strong enough to make this promise. It should be extremely flexible and interoperate with tools in a way which is highly acceptable to tool vendors.

The technology used for the tool-net should be open rather than proprietary and not depend on a single point of failure vendor.

<table>
<thead>
<tr>
<th>Version</th>
<th>Status</th>
<th>Date</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Final</td>
<td>2012-06-04</td>
<td>22</td>
</tr>
</tbody>
</table>
4 Tools interoperability concepts

The preceding discussions bring up the following requirements of the tool-net:

- Use a sound stable and widely accepted standardized technology that will persist and survive the long period expected of the SoS lifecycle
- Built on the concept of tools evolution as well as work methods along the tool-net life-span as well as the life-span of an SoS designed on the tool-net.

The requirements of the tool-net by the different work packages and users sums up to the following list:

- Tools interoperability
- Distributed interoperability of tools
- Collaborative work among multiple partners and over wide geographic spans
- Work with certain tools of preference by the immediate intended users of the tool-net in the project
- Support multiple domains of operation such as requirements, design, testing, change management and do on.

4.1 Foundation standards of the tool-net

To answer the first bullet above, and as already stated in the project DOW – DANSE will be based on sound web and internet technologies which have shown that they are stable, widely accepted, and sound technologically as proven by the ever expanding, increasing and improving web network.

Specifically, the semantic web is considered today the means to dress the seemingly untamed and unorganized web into a semantically oriented ocean of information. DOD has recently made a statement of belief in this technology [ref] that we ought to take into heart. The actual technology we adopt are as follows:

B-a) The use of **RDF** [3] to represent information resources in the tool-net and exchange that information among tools and tool-net services.

B-b) **OWL** [26] language to describe meta-data of models and by which meta-data can be related to each other and enable semantic interoperability among tools, each basically working with its own internal language and representation.

B-c) **W3C Linked data** [27] as the means to connect relevant pieces of information resources among tools and via the tool-net services.

B-d) The notion of a resource as a web resource -- an entity uniquely defined over an infinitely large name space domain, and such which can be addressed over the internet to provide additional information.

B-e) The idea that is the base of an open-world-assumption [33] -- that there are no predefined schemas of information and that facts can be concluded for good or for bad - only based on other facts.


B-g) More particular to the engineering design tools – we will adopt the evolving standard (which is based on the above technologies) of the Open Services for Lifecycle Collaboration – **OSLC** [8]. That standard specification already opens up tools and applications to interoperate, harnessing the internet and web technologies to create an agreed way of exchanging data in an open world where sometimes, not all the information about a resource is known at one point in time, but which may become available as the tools progress and adopt more features, evolving in their own pace and gaining value from joining the interoperability space.

B-h) Use of the **SPARQL** [4] query language for working with RDF model data, and related query syntaxes and tools.
Based on these technologies, to manage the model data by the DANSE collaboration platform, working with the design tools and applications, will require to apply some new technologies – in as limited way as possible to not create a problem of maintaining new proprietary implied technology and tools:

I-a) A mechanism to ensure protection of ownership by partnering corporate members in the SoS, each developing their own constituents (of leading/managing the development and evolution of such constituents), while sharing only what is required and only with specific target partners to carry out the interoperability and integration job into the SoS.

I-b) A distributed platform to manage the shared modelling data so that corporate partners can each work in their own environment, while sharing according to the above protection of ownership, and having the distributed access across the globe.

I-c) A mechanism that will allow model data of collaborating tools to be shared in that platform, by sharing a common semantic via meta-models and relation among these models that can be processed and create meaningful links among design resources in these models so that semantically related elements can be accessed and follow by partners. Such a mechanism, which in the SPRINT project 2.5.3) is called “semantic mediation” will be necessary in DANSE as well.

4.1.1 Support traceability of users’ requirements

We hereby present Fehler! Verweisquelle konnte nicht gefunden werden, extracted from DANSE requirements repository, where requirements relevant to the tool-net can be traced to the technologies, basic and implied, listed above. DANSE requirements repository is an internal document, shared between DANSE partners. We used a snapshot of this repository at T0+6. For more details, please refer to deliverable D3.1 [31], which explains DANSE requirements management process. Though the elicitation of DANSE technical requirements is scheduled to continue over the next months, we assume that the main orientations shown in the table below will not be questioned, hence provide a meaningful basis for technology traceability.

Table 1: Traceability of users’ requirements to basic and implied technologies

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Base technologies</th>
<th>Implied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description (simplified)</td>
<td>B-a</td>
<td>B-b</td>
</tr>
<tr>
<td>UN-05-001</td>
<td>Handle legacy and new LoB components</td>
<td></td>
</tr>
<tr>
<td>UN-05-002</td>
<td>Improve integration of methodology and tools</td>
<td>X</td>
</tr>
<tr>
<td>UN-11-001</td>
<td>Incorporate simple and reliable tools</td>
<td></td>
</tr>
<tr>
<td>UN-11-003</td>
<td>Provide tool platform for SoS engineering through industrial use cases</td>
<td></td>
</tr>
<tr>
<td>UN-11-005</td>
<td>Moving from SoS “picturing” to SoS “understanding”</td>
<td>X</td>
</tr>
<tr>
<td>UN-06-004</td>
<td>Provide traceability between top and bottom levels</td>
<td>X</td>
</tr>
<tr>
<td>UN-05-003</td>
<td>Synchronize global SoS goals with local components’ goals</td>
<td>X</td>
</tr>
<tr>
<td>UN-00-001</td>
<td>Improve collaboration in SoS development</td>
<td>X</td>
</tr>
<tr>
<td>UN-10-006</td>
<td>Preserve developers’ IP</td>
<td>X</td>
</tr>
<tr>
<td>UN-05-005, UN-05-006, UN-06-005</td>
<td>Identify changes (dynamicy) in constituents</td>
<td>X</td>
</tr>
<tr>
<td>UN-04-002</td>
<td>Validate/verify cross system domains and interfaces in dynamic SoS</td>
<td>X</td>
</tr>
<tr>
<td>UN-11-008</td>
<td>Discover useful abstractions of SoS</td>
<td>X</td>
</tr>
</tbody>
</table>
4.2 Requirements toward the participating tools

Participating tools in the tools-net must abide with interoperability standards which are reflected in the technologies adopted by the tool-net, and which will also allow to apply the implied technologies to be developed in the project. Moreover, complying with these requirements will enable the tool-net to provide answers to the users requirements as seen in the traceability table above.

A tool SHOULD comply with OSLC. Yet, we allow here some levels of flexibility, taking into account that some legacy tools may not be able to adapt fully with the new standards. Therefore we distinguish Client compliancy from full Server compliancy.

- **OSLC Client compliancy** – represents the ability to actively engage an OSLC server to share model information of the tool. This includes the management of its resources although these resources cannot be accessed directly from the tool as OSLC dictates, the resources can still be consider as such. These resources are shares as such with the tool-net platform via export (which we also term publish) and import activities, both driven and activated on the tool by their user, which therefore requires some UI extensions as summarized in these sub-bullets:
  - Manages its resources
    - Can **publish** contents into the DANSE shared common-semantic platform
    - Can **import** contents from the DANSE shared common-semantic platform
    - Optional UI extensions
    - Can index its resources in the DANSE platform over the internet. To be able to do this indexing, means that resources of the tools store their URIs [27] so they can be linked over the tool-net platform.
  - Mechanisms for sharing models as clients are also quite flexible and at their base, these can be file based, with more advanced capabilities desired such as the RESTful protocol, and better off the OSLC compliant protocol (which is also based on RESTful).
    - File-based proxy server
    - Direct OSLC protocol
    - Direct RESTful protocol

- **OSLC Server compliancy** – this is certainly a capability which modern tools will have, being servers and providing access to resources (model resources) the tools manage. There are several levels of compliancy for servers as well and being clients is one of them (same as above). When fully implementing all the OSLC recommendation capabilities, the integration into the DANSE tool-net will result with very powerful possibilities, as also listed in the following bullets.
  - OSLC management of its resources
    - Can respond to OSLC “crawler”
    - Can index its resources in the DANSE platform over the internet
    - Can do all that a client can do.
  - Mechanisms:
    - OSLC protocols

| UN-10-011 | Incremental formalization of requirements | X | X | X |
| UN-10-008 | Feeding behaviour from operation | X | X | X | X |
| UN-10-010 | SoS engineering environment over long time scales | X | X | X | X | X |
### 4.3 Interoperability concepts

The basic interoperability concept of the tool-net is the use of the OSLC protocol and specifications [8]. In a nutshell, OSLC defines clients and servers which exchange data in RESTful protocol, and whose payload are RDF resources. These resources represent model elements. OSLC servers can respond to RESTful requests with services such as factory to create new resources, query to pick relevant resources to a certain criteria, descriptions of resources which define the “shape” of a resource and graphical renditions of the resources that can be displayed with a web browser. With these services, tools can exchange data, maintain cross links to resources in other tools and even create diagrams displaying resources of other tools together with their own resource renditions and the links relating resources of all resources, local as well as foreign.

OSLC identify several domains of tools in which resources may be of different types, serving the lifecycle of products. This includes requirement management (OSLC/RM), architecture management (OSLC/AM), change management (OSLC/CM) and so forth. Tools in each of these domains will maintain closely corresponding resources and by exposing server interfaces to manage these resources and establish links between them – the collection of tools will create a holistic view of the product lifecycle. That, compared with the disconnected collection of views and representations is a great advantage to users.

The integration that OSLC facilitate is actually a point to point integration. Every tool can interact with any of the other tools accessible to it. The inter-links among resources of different tools need to be maintained by some entity. Since every tool can serve that entity, the result will be that this info is replicated and probably is also incomplete and partially done, as well as a problem in establishing it. The following Figure 11 dramatizes this problem as a mess of tools (on the left) each trying to “understand” all the other tools relevant to it.

![Figure 11: Point to point vs. "hub and spoke".](image)

With the server in the middle, each tool needs only to exchange information with one target which will create and maintain the holistic view of the system under design, where each tool contributing info only to this entity.

Each of the links with the central entity is an OSLC link, working according to the OSLC protocol carrying OSLC payloads, i.e., RDFs.

The model data exchanged with the server is not anonymous, nor is it solely categorized by the main OSLC domains. In DANSE, we follow the SPRINT approach, in which there are ontologies governing the meaning and semantics of the model elements maintain by each tool. Ontologies in the OWL [26] specification (also an RDF syntax) define the content in each tool.

The relation of tools, models, ontologies and the rules which dictate how resources in different models and tools are related to other resources in other tools are schematize in the next Figure 12.
Figure 12: models, relations, ontologies and rules.

In this figure, tools and applications on the top may be working on a common project and expose models as "user documents" with a related "reference" model in which some aspects of the models are maintained. There are relations between the models which are maintained against the reference model. These relations are depicted in this diagram as rules. It is possible to apply these rules to extract such relations from the ontologies layer in which ontologies for each tool as well as for the reference model provide in combination with "semantic mediation" rules the ability to create links that connect the individual models into a single holistic view of the project.

The roles which tools and applications play within the OSLC specifications are also depicted in the following Figure 13.

- Clients may have no repository
- Clients cannot respond to queries
- Clients are not servers
- BUT: Clients can participate in OSLC

Figure 13: OSLC servers and clients.
The OSLC client here can work with other servers being tools, but also they can be the collaboration server in the middle of Figure 11. Yet, fully active OSLC servers can also work with each other.

The full possible configurations of tools, applications and the DANSE tool-chain platform will be disclosed in the architecture chapter of this document (6 “Architecture principles of the Tool-Net”).

4.4 Expectations of tools

Tools for the tool-net must be compiled into an evaluated list according to their provision of the capabilities required of them above, but also following a discussion among users and tools/application providers of the DANSE community per their need within the DANSE engineering ecosystem.

The following is a template to be applied to tools as a means to identify their applicability. For that, we need to answer questions related to:

- How are the tools involved in the test-cases? To answer this question, users should describe the following details:
  - Detail of data flow among the tools – a design scenario.
  - Details of the artifacts produced and consumed by each tool.
  - Alternative tools to consider for the same purposes.

Important points to consider in light of the DANSE methodology discussion above (2.1 “SoS methodology concepts – from WP4”) per each tool are the following:

- DANSE tool-net is not used to design constituent products “from A to Z”, rather – it is the SoS design and constituent interfaces (for which there are “contracts”). Tools for that purpose can work in any of the engineering environments owned by the constituent owners, and should be “flowed” to the DANSE ecosystem at a high level rather than at their intimate engineering details. Such details are most likely to be part of the owner and developer intellectual property (IP) as well as improper -for its details depth- to be considered at the SoS level.
- Complex system design methodologies and tool-nets are more appropriate at the constituent level rather than in the SoS DANSE level. So, consider using the results of SPRINT [7] to design and manufacture constituent products.

Therefore, not all tools are equally mandatory. To include a tool – user must show that it is critical for SoS-level design.

Nevertheless, the IAI user have proposed the following table of tools, which serves in this document to be a basis for discussing tools into the DANSE tool-net.

<table>
<thead>
<tr>
<th>User</th>
<th>Tool name</th>
<th>Tool Vendor</th>
<th>Category</th>
<th>Time needed</th>
<th>Use case</th>
<th>Discipline</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAI</td>
<td>DOORS</td>
<td>IBM</td>
<td>Requirements</td>
<td>Start</td>
<td>Water</td>
<td>Cross discipline</td>
</tr>
<tr>
<td>IAI</td>
<td>System Architec</td>
<td>IBM</td>
<td>Design</td>
<td>Start</td>
<td>Water</td>
<td>System</td>
</tr>
<tr>
<td>IAI</td>
<td>?Focal Point?</td>
<td>IBM</td>
<td>Design</td>
<td>Start</td>
<td>Water</td>
<td>System</td>
</tr>
<tr>
<td>IAI</td>
<td>?Simulink?</td>
<td>Mathworks</td>
<td>Design</td>
<td>Midway</td>
<td>Water</td>
<td>System</td>
</tr>
<tr>
<td>IAI</td>
<td>Rhapsody</td>
<td>IBM</td>
<td>Design</td>
<td>Start</td>
<td>Water</td>
<td>SW</td>
</tr>
<tr>
<td>IAI</td>
<td>Design Manager</td>
<td>IBM</td>
<td>Design</td>
<td>Start</td>
<td>Water</td>
<td>Cross discipline</td>
</tr>
<tr>
<td>IAI</td>
<td>?ICD Tool?</td>
<td>ADI</td>
<td>Design</td>
<td>Start</td>
<td>Water</td>
<td>Cross discipline</td>
</tr>
<tr>
<td>IAI</td>
<td>?UGS &amp; Team Center?</td>
<td>Siemens</td>
<td>Design</td>
<td>Midway</td>
<td>Water</td>
<td>Mech.</td>
</tr>
</tbody>
</table>

Table 2: Tools proposition by IAI, not discussed yet.
<table>
<thead>
<tr>
<th>IAI</th>
<th>?Blend?</th>
<th>Microsoft</th>
<th>Design</th>
<th>Midway</th>
<th>Water</th>
<th>MMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAI</td>
<td>?Quality Center?</td>
<td>HP</td>
<td>Testing</td>
<td>Midway</td>
<td>Water</td>
<td>Cross discipline</td>
</tr>
<tr>
<td>IAI</td>
<td>Rational Team Concert</td>
<td>IBM</td>
<td>Task mgmt &amp; CM</td>
<td>Midway</td>
<td>Water</td>
<td>Cross discipline</td>
</tr>
<tr>
<td>IAI</td>
<td>RELM</td>
<td>IBM</td>
<td>Cross tool activities</td>
<td>Midway</td>
<td>Water</td>
<td>Cross discipline</td>
</tr>
<tr>
<td>IAI</td>
<td>MANTRA?</td>
<td>IAI</td>
<td>CM</td>
<td>Midway</td>
<td>Water</td>
<td>Cross discipline</td>
</tr>
</tbody>
</table>
5 Design modelling semantic integration concepts

5.1 The “common semantic” concept

In DANSE, a SoS is a single coherent entity and the tool-net should provide a holistic view of this entity, which is aligned with the working methodology for SoS engineering. The means to create that coherency is to enable information to flow over all components of a SoS at all their levels and within all their possible interactions.

The concept of “common semantic” is an essential foundation of this approach, and the tools interoperability and the consolidation and federation of their modelling data should be compliant with that concept. The practical meaning of this concept is that mechanisms in the tool-net will be developed to relate the semantics of the different contributors to each other. One such mechanism is the OSLC specifications; another is the use of semantic mediation.

With these mechanisms, it should be possible to apply additional technologies such as the DANSE work packages will invent, develop and apply: methodologies, architectures, modelling languages, analysis and optimizations and simulations.

Each and all of these technologies will be modelled in a common representation and according to a selected specification as detailed in 4.1 "Foundation standards of the tool-net". These foundation standards will serve also the mechanism to define common semantics and apply that to the models in each and every one of the DANSE technologies.

Some principles of the common semantic concept:

1. It is not assumed that there is one "lingua franca" that can serve all modelling aspects of SoS, yet there are several common concepts that are shared by multiple modelling tools.
2. Identifying such common concepts can lead to definition of several common semantic sets.
3. The common semantic sets can be governed by rules that will be used to automate the relation among tools for two purposes:
   a. Transforming model elements from one tool to another, or more precisely, creating and updating model elements in one tool based on model elements in another tool
   b. Creating links among model elements of different tools.
4. An instantiation of common semantic model elements will serve as a reference model that can be used to anchor models from different tools in a meaningful way to the end user.

In the following sections we will elaborate on the semantic mediation concept as is also developed in the SPRINT project and which is a candidate to be adopted in DANSE for similar, yet extended use cases.

5.2 Approaches to meet a common semantic among diverse tools

5.2.1 Semantic mediation

The basic semantic mediation allows tools using different languages to share the common semantic of their languages at some higher abstraction that can indeed commonly abstract all these languages. That is depicted in the following Figure 14, where several tools are shown – in this case the IBM Rational Rhapsody® and the MathCore MathModelica® tools. The common semantic has been defined based on SysML [28], limited to a basic structure modelling of classes and instances of classes as parts of larger elements, which is termed “Basic Structured Ontology” or BSO for short.
Figure 14: Exchanging models from different tools via the common semantic model.

Taking for example the flow of data from Rhapsody to the repository, this is depicted in the next Figure 15, showing a sequence of activities following the interaction of the sequence diagram in Figure 17. That sequence diagram is the implementation of the tool-specific protocol, which is shown in the architecture patterns diagram (see further on in Figure 19) as items (7) (in the tool) and (1) (in the server).

Figure 15: The flow of model data into tools-net server from Rhapsody
5.2.2 Typical Tool-Specific tools-net Flows

Three flows are described in Figure 17:

1. Initial model posting to the server
2. Getting that model by another tool to initiate its own model
3. Posting an update to the SII model by that last tool.
Conceptual and architecture principles of SoS design and semantic interoperability of systems platform and the SoS design Tool-Net

Figure 17: Tool-specific RESTful protocol between tools-net server and tools.

(1) The first flow is the posting of new model data into the server from the MathModelica® tool. That is also the flow shown in Figure 15 - for the Rhapsody tool though. In the initial posting of a model data from the tool, an RDF model (instance of the tool’s taxonomy and meta-model) is posted into the server, which in turn converts it to a BSO model and stores the new resources in the repository. In response to a successful post, a list of matching URIs – those in the repository and those coming from the tool – are returned for the tool to store as annotations (or other means) in the tool for follow up activities.

(2) The second flow above is where a second tool (in this case – Rhapsody) is fetching the BSO model into its internal modelling language. Being a new model in the tool, it is fetched via the GET API of...

<table>
<thead>
<tr>
<th>Version</th>
<th>Status</th>
<th>Date</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Final</td>
<td>2012-06-04</td>
<td>33</td>
</tr>
</tbody>
</table>
the HTTP protocol (on top of which the RESTful protocol is applied). The returned RDF model is queried from the RDF store of the tools-net server, converted by the Rhapsody plug-in in the server to a Rhapsody DRF representation (or any representation preferred by the tool), and returned to the tool. In this particular case, Rhapsody would generate comparable elements in its internal modelling language for the elements obtained from the server, and also maintain an association between the URIs internally generated in the tool and those coming from the server.

The third flow is an update of an existing shared model in a tool with the server, where some additional elements are modelled in the tool, which have not been previously shared. The tool posts to the server very much like it did in the first flow, yet at this time, a target elements in the common model (BSO model) is already identified so that only the new elements are converted and stored in the server RDF, associated with the older identified elements which are not recreated, nor duplicated.

5.3 Modelling and tools domains relevant for DANSE

This section depends on the progress in other technical work packages as well as use cases. Domains as can be identified for each of the major work packages and as described in the introductory chapter are:

1. Architecture patterns modelling and application. That seems to be a mechanism to model patterns and how to apply these to design models.

2. Modelling languages relate to such tools as are commonly used to model SE designs.

3. Analysis, optimization and simulation applications are a domain that uses designs such as done by modelling languages. Such applications may require that models that are analyzed can indeed provide the required information and attributes as needed for the job. Results of these applications are also information entities that should be incorporated within the SoS modelling environment.

4. Methodologies of work that dictate the process of design and the management of the SoS life cycle. Modelling methodologies is a domain to be developed and methods of applying it to the engineering echo-system must be provided.

5. Handling dynamicity and evolution in the SoS through modelling must be defined and articulated in terms of modelling or model changes.

6. Change management modelling may be one possible articulation of handling dynamicity of item #5 above.

Each of these domains must produce requirements for services in the tool-net which must be weighted against the capabilities of the tool-net as described and traced in 4.1.1 “Support traceability of users’ requirements” above.
6 Architecture principles of the Tool-Net

The tool-net architecture is a “hub and spoke” architecture (see Figure 11, page 7), where the hub is the tool-net platform and is distributed. That architecture is reusing the SPRINT integration platform as shown in Figure 7, page 18. This architecture depicts a distribution of the hub over multiple internet spaces, each owned by a partner. The distribution concept is dealt in more detail below in the “Distribution concepts” section.

Tools and application interact with the hub platform in multiple ways we call patterns which allow a variety of integration methods of tools, applications with the hub and among them.

6.1 The concept of IP protection

An important principle of the tool-net is the protection of partners’ IPs. That concept is supported via the architecture of the hub servers and the no-direct links between modelling tools to share their information as is the case which OSLC supports. A direct access among tools of different vendors is not customary possible due to intranet networks isolation. A tool in one corporate operates within the intranet boundaries of that corporate and cannot connect with a tool in another intranet. This situation is described in the following Figure 18.

![Figure 18: Servers interoperability over the internet (as done in SPRINT)](image)

DANSE tool-net relies on a multi domain architecture where hub servers owned by each partner resides in its "yellow zone" domain. Servers in that domain are allowed to be accessed by work stations and computers in the intranet of that partner. At the same time, different machines in different yellow-zones can communicate over the open internet within certain, well protected and secured protocols.

Within this technology which allows exchange of data among hub servers over multiple yellow zone machines, DANSE will apply access isolation among the partners which controls what and who can see what and from where in the logical modelling space of DANSE. The data isolation is applied through two mechanisms. Firstly, the modelling tools of each partner publish data only to their own hub servers in each yellow zone domain, but not all the modelling data, but that data which is required to communicate with partners. That will never contain details of designs.
Secondly, the tools can access data from other partners only through their own hub server in their own yellow zone, but again, by having this hub server serve them as a proxy to access other hub servers in other yellow zone over the internet. In doing that, only data which is permitted for tools of that partners is accessed and obtained from the other partners.

Modelling data which represents highly valued IP should not be published to the yellow zone hub servers, and the less critical data which is published, is bound to access isolation rules that further prevent partners from seeing details they should not share. This is the responsibility of a hub server when approached by another hub server requesting data so that nothing is sent over unless permitted.

The technology to permit that is an important service of the tool-net platforms, yet it is not expected that DANSE will provide a fully proofed solution, but a guidance on how that could be done in the real world. That is so since the real world will be very reluctant to rely on the second data isolation mechanism and will firstly trust only the first mechanism of not publishing any details at all to the hub server on the yellow zone. Moreover, it is more likely that in very sensitive companies, the fear of the internet is so high that any access into a yellow zone machines will be forbidden in the general case.

It is the DANSE assumption that in many commercial situations where the DANSE technology and solutions will be applicable, such a high level of sensitivity will not exist and that the interoperability level among partners will indeed be significant so that the isolation mechanisms described here will suffice.

### 6.2 Architecture patterns of the Tools-net

There are many different architecture patterns for tools, applications and the DANSE tools-net platform. The possible interoperability patterns are depicted in this Figure 19.

![Architecture patterns of the tool-net](image)

**Figure 19:** Architecture patterns of the tool-net

The architecture deals with the following patterns:

1. Legacy tools - are such tools in which it is not possible to extend with export facilities and therefore these tools need a companion agent that can do that for them, by having an access to their repository or some other API.
2. OSLC Client is a generic tool, or one adapted for a specific tool (similar also to the agent working with legacy tools), and which can provide access and update to modeling data in the SII via the OSLC protocol supported by the SII platform.
3. Applications and OSLC tools – are the non-legacy tools which can be extended with UI and OSLC functionality and interact in this way with the SII platform.

4. SII agents and tools – are such developments which use the programmable and extensible SII platform being DM or Jazz, to develop applications and tools that can provide added value by operating on the stored data within the platform.

Details of this figure as per the numbers and letter tags is as follows where we numbered components in the tools and in the collaborative SII platform, and marked with letters the different tools and application patterns. The components (1)-(4) and (9) are extensions of the JTS platform, whereas those numbered (5)-(8) are parts of the tools and applications. Tools and applications marked (A)(E) are external to the platform, while 0 is an application built within the platform as a Jazz application. The other parts (in dark blue) in this figure are existing tools and applications that are used within the DANSE tool-net ecosystem.

### 6.2.1 Components

(1) Tool Specific. An extension in the RESTful internet interface of the SII which is developed specifically for different tools and which allows massive exchange of model data between the tool/application and the platform. The service implementation of this interface can do the initial mediation of tool data as it flows into the repository (see 5.2.1 “Semantic mediation”) and the inverse mediation of models in the repository to specific models in the tools and applications.

(2) Semantic Mediation. A framework of plug-ins to the platform each of which performs a certain step in the composition of transformations between concrete tools' models and a multi-level structure of common semantic models of various domains as will be describe in chapter 5.2, “Approaches to meet a common semantic among diverse tools” in this document.

Repository Management. A layer within the platform to manage specific capabilities such as access control to data, distributed federation of model data over multiple nodes (see for more in section 0 "

(3) Repository Management"), and the change management of models shared by the tools. Of all these capabilities, the distribution and federation of the data is most critical and will be embedded within the repository access software layer between mediation and the RDF repository. The additional important capabilities will be of a secondary urgency for the project and may also be provided by the Jazz platform as it evolves during the project.

(4) UI Extension on the platform. The additional SPRINT services to the Jazz platform may need GUI extensions that will be integrated within the GUI framework of the platform as GUI plug-ins (see chapter 6.5.2 “Plug-in for Extending the Web GUI”). We can anticipate some management services of the repository and its configuration over a distributed environment as such which need some GUI extensions as well.

(5) UI and its possible extensions for the integration of tools with the platform. Tools, agents, and applications integrated into the SPRINT SII ecosystem need some UI extensions over their native GUI framework, or on top of that, which will support the specific operations of end-users taking advantages of the added value offered by the SII platform.

(6) OSLC client for tools and applications. OSLC client functionality is a subset of the full OSLC server capabilities and is a minimal entry-point requirement of tools in the SPRINT tool-net ecosystem. An OSLC client function allows the tool to work against the OSLC Server interface of the SII. With this interface, model elements in the SII repository can be queried, read, modifies, created or deleted by tools. Note that this interface can also work through the semantic-mediation layer as much as it can be done through the tool-specific interface (see 5.2.2 "Typical Tool-Specific tools-net Flows" and (7) in this list)

(7) Tool specific (or application specific) interface that may not be OSLC compliant, yet it is still RESTful compliant. This is a complementary interface in the tools to the interface on the server (see 6.2.1), allowing direct and massive exchange of models between tools and the SII repository through the semantic mediation layer.

(8) OSLC server is a fully developed OSLC compliance of a tool that can provide full access to content of the tool. When a tools is OSLC server, its content can be interrogated and accessed automatically by the SII server (see next point (9)) with the tool being agnostic to the SII server altogether. That is a great advantage for integrating tools into the SPRINT ecosystem.
(9) OSLC crawler is a powerful capability of the SII that is not mandatory, but a very convenient way to integrate OSLC compliant tools into the SPRINT tool-net ecosystem. As the OSLC protocol - when fully implemented on a server - is fully self-descriptive, a crawler can find out the content of a tool as much as web crawlers can detect contents over the internet.

6.2.2 Tools and applications:

(A) Data extractor and importer. This is an application which helps to integrate a legacy tool into the SII tool-net ecosystem, w/out making any changes to the tool. Having its own OSLC client interface and the tool-specific interface, a UI, and a full access to the internal data of a tool – this application can communicate contents of the tool projects with the SII. An example for that is the Java API of Rhapsody which allows developing such application which is able to work with Rhapsody project repositories w/out doing any changes to the Rhapsody product.

(B) Legacy tool is a non-OSLC compliant tool which can be integrated using the Data Extractor-Importer. In this variation, the tool is extended (as most of the tools today are capable of) with the needed functionality that will allow them to integrate with the SII eco-system. In fact, the same interfaces and UI extensions required for the stand-alone application described in (A) above are also required to be extended into the legacy tool. Again, our example with Rhapsody can be performed by applying the added functionality via a Rhapsody plug-in, which extends the Rhapsody functionality via a profiles mechanism that uses the Java API to access the tool’s data and interface it with the SII eco-system.

(C) Application interacts with the SII in order to perform some value-added function over the model data stored in the SII repository. Examples of such applications are simulators and analysers, test generators and optimizers and the like. The application needs an OSLC client interface to obtain model data from SII at a minimum. However although not shown in this diagram, an application may also have its own specific interface (RESTful though) by which massive model data can be obtained in its own specific representation – much the same as it is done for tools (i.e., (1) and (7) above).

(D) OSLC tool is a contemporary tool that is OSLC compliant, meaning that it has full OSLC server capabilities. An advanced capability of SII to “crawl” the network can take advantage of such tools and work with them w/out the tool being aware of the SII eco-system altogether.

(E) OSLC client is a generic application that can execute OSLC client functions for querying, reading, modifying, creating and deleting OSLC resources of the SII. The OSLC client is also an OSLC sample project which can be used to customise specific clients for different tools as much as the “Data Extractor – Importer” in (A) above.

Internal SII application takes advantage of the SII platform itself to support extensions such as to perform specific value-added functions over the SII repository. For instance, an impact analysis application, or a validation application which have their own UI extensions through the UI plug-in framework of the SII platform and which work directly with the content of the SII repository.

OSLC crawlers are applications working over the SII platform and which can use the OSLC protocol to work with full OSLC server tools so that export/import functionality can be provided w/out changing the OSLC tool. Being OSLC is a major step in providing interoperability to tools, which can be utilized by such agents to integrate its data into the SII platform and enjoy the powerful semantic mediation on it.

6.3 The platform

The DANSE collaboration platform is Jazz [1], which is depicted in the diagram of Figure 20.
The Jazz Team Server (JTS) component can work with multiple components over the internet. A distinguished element in this server is the Jazz Foundation Server (JFS) which provides an RDF repository as well as additional services including full text search, indexing, RDF query and so on.

Tools in this context are Jazz applications that can be developed by any vendor and there are several such applications developed by IBM. For instance, the Rational Change Management (RCM®) application which manages projects with tasks, bug reports and so on, or the Rational Team Server (RTS®) in which code and model versioning service is provided. A specific component which will be used to implement the DANSE tool-net server is the Designs Management (DM).

The DM will be used to define domains of modeling to serve the common semantics discusses above and will host the services required by the tools in the tool-net. The semantic mediation is one of the services on this server, the distribution of this service over multiple servers is also discussed in the next section and is another component used internally in this server. We also identify the possible need to implement additional services over Jazz and DM. Both types of services which in the architecture patterns (Section 6.2) is identified as “Agent” can be done by extending the plugin mechanism of the DM and the Jazz – both defined as an open platform and will also be described briefly in this document.

6.3.1 Agents

Agents are generic adapters for content providing tools that is depicted in the architecture block diagram in Figure 19 as “OSLC crawler” (marked as (9)) which works over the internet (actually the intranet) with an OSLC server (marked as (8)). This is an advanced functionality where the server may pull model data from a tool rather than the tool pushing into the server as described above.

Looking back in the data flow through Figure 15, the initial mediation stage between tool's data format and structure to the OSLC RDF is performed via this OSLC query and retrieve standard OSLC services. The resulting extracted model data can now be pushed through the mediation stages same as in the push case.

The advantages of this approach are that the tool is SII-agnostic, although it is OSLC-aware. It is expected that as the OSLC initiative gains wider acceptance, this can become the preferred integration method for tools with the server.

6.3.2 Internal Applications

Just like the semantic mediation service and agents, an application within the server can access the model data and perform various value-added processing on it much as any other external tool. Such application can be an entire modelling tool, or a simulator, and so on. That is identified in the architecture block diagram (Figure 19) as (F).
Working within the Jazz server or the DM is a mere convenience as all the information available within the is also available via the OSLC API to any external application. The common thing to the semantic mediation service, agents and applications is that they all extend a plug-in framework of the Jazz or DM. These are different plug-in frameworks, which differ in what services can a developer use. That framework allows to extend the back-end for access to the data repository as well as GUI extension points so that a web interface to users can be exploited based on a rich framework for model graphs rendering and manipulation on a supported web browser.

Internal application examples are:

1. Relations management – an application which maintains the index of external resources owned by participating tools, and which are related among themselves as well as with the common model element resources owned by SII (as shown in Figure 16).

2. Validation – an application which can apply validation rules at various levels and areas. For instance, a methodology validation which ensures that all tools indeed work according to some methodology, such as ECSAM [35]. Validation and verification of the semantics of a design – perhaps using contracts – is also possible as an internal application.

3. Various visualizations of the server model contents over the web in the distributed environment, for managing the complex data sharing of DANSE projects.

4. Mediation configuration tool – a tool to configure the paths to be taken by different streams of model data published from and imported into different external tools.

Mediation rules work-bench is a GUI intensive application where transformation rules of mediation stages can be authored and maintained.

### 6.4 Distribution concepts

The following Figure 21 is a diagram of tools interoperability through the DANSE tools-net server. It demonstrates two scenarios. In one scenario, tools exchange data through the same server while the second one demonstrate how it will be done over a remote tool.

![Figure 21: Tools interoperability through the DANSE tools-net servers](image)

There is no distinction at the tools level between the two cases and the federation of model data over the two servers is transparent to the tools. Important concept is that tools do not “speak” directly with each other, and do not work in synchrony.

In fact, a tool may export (or publish) model data to the tool-net server and import model data at any time unaware when and whether that data is used by another tool.

This concept is driven by both the distribution characteristic of the tools-net, and the multi partner environment which must protect partners IP rights as discussed in 6.1 above.
A specific scenario is demonstrated - and has been implemented over Jazz DM for the SPRINT project – in this next Figure 22, where two servers serving two partners in their own yellow-zone internet. Several tools are used here, which publish model data into the server in which the model elements and relations amongst them are maintained. A view into the model elements, revealing their owner tool and owner partner is given in Figure 24 below.

**Figure 22:** Specific configuration of a distributed collaboration among two partners over the internet

The role of servers, tools and the areas where they operate over the internet are all shown in Figure 22, demonstrating the concepts we discuss here. Elaborating on this distribution is again demonstrated in Figure 23 which is yet another generalization of the diagram used for SPRINT in Figure 9. The generalization shows that a server may be reused to server multiple distributed projects showing how a partner may be engaged in more than one collaborative project with different partners.

**Figure 23:** A generalized collaboration scenario depicting the management of multiple distinct projects
This general power of the DANSE tools-net distribution and collaboration relies entirely on the concepts of tools isolation, data isolation and access rights and the federation of data over multiple servers via secured communication among Jazz servers.

Cross organization collaboration

Figure 24: GUI view of a distributed modeling data over corporates and tools

6.5 Extensibility of the platform

The Designs Manager DM platform supports advanced data management services that can and should be applied to the server repository management. The tools and applications working with this server must be adapted to work with these additional features as indicated in the next Table 3.

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OAUTH Authentication</td>
<td>Provides OAuth client utility for authenticating server access for the tool and internal-project against SII core security model. See [25].</td>
</tr>
<tr>
<td>2</td>
<td>Version Control</td>
<td>Negotiates with the SII server for the next version information to enrich resources data.</td>
</tr>
<tr>
<td>3</td>
<td>Target Project UOC Manager</td>
<td>Negotiates with the SII server for target project to publish data for.</td>
</tr>
</tbody>
</table>
The Jazz DM server consists of the basic Jazz Foundation Server platform – the JFS, which implements the basic framework of all Jazz applications, including the DM. What DM brings is a rich and powerful modelling management capability in the repository back-end of Jazz and the web GUI for model graphs management. DANSE will use these advanced features rather than just the basic foundation capabilities. So, the DANSE plug-ins will be built on top of the enhanced DM framework.

### 6.5.1 Plug-In Components

We identified in the previous chapters plug-in components such as the semantic mediation service, internal applications, crawling agents, mediation configuration and authoring work-benches of mediation rules. The following sections discuss the plug-in mechanism support provided for DANSE. A detailed reference for the API is part of the SDK for the java projects developing the plug-ins.

The tool agent plug-in project extends DM front service as follows (only an example):

```xml
<extension point="com.ibm.xtools.rmps.frontservice.contentProvider">
  <contentProvider>
    <name="My Tool Generic Content Provider"/>
    <uri="http://my.company/MyTool/1.0/"/>
    <version="1.0.0"/>
  </contentProvider>
  <publisher>
    <contentProvider="http://my.company/MyTool/1.0/"/>
    <id="application/x-mytool-model"/>
    <name="My Tool Content Provider"/>
  </publisher>
</extension>
```

### 6.5.2 Plug-in for Extending the Web GUI

This section details the GUI extension API in some fine level. To extend the web GUI with the plug-in mechanism, the DiagramProvider class instance must be implemented, extending the following set of API.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getImage()</td>
<td>Returns a typed image object formatted with one of the requested formats (e.g., image/png, image/jpg) for the given resourceUri. A null is returned if there is no diagram resource associated with this resource, and an exception is thrown if there is any invoking or accessing other services through the jfsClient.</td>
</tr>
<tr>
<td>jfsClient</td>
<td>An authenticated reference to a JfsClient object that the implementations can use to invoke SII and JFS services.</td>
</tr>
<tr>
<td>diagramUri</td>
<td>The URI of the architecture resource to generate an image for it.</td>
</tr>
<tr>
<td>acceptableFormats</td>
<td>An array of acceptable mime types (e.g. image/png, image/jpg) ordered by preferred types</td>
</tr>
</tbody>
</table>
### getTileDescriptor ()

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITypedImageDescriptor</td>
<td>Returns a tiled image descriptor that is an object that indicates how an image is tiled. This object indicates the overall width and height of the entire image, and contains a list of ids that are used to reference specific tiles in the overall image. If a content provider supports tile-ing then it will provide an implementation to the getTiledImage() method that responds with the tile for the given diagramUri and tile Id. Tile Ids will be URL encoded and passed back to the service by the client as a query parameter when the client is ready to render that particular tile.</td>
</tr>
</tbody>
</table>

### jfsClient

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IJfsClient</td>
<td>An authenticated reference to a IJfsClient object that implementations can use to invoke SII and JFS services.</td>
</tr>
</tbody>
</table>

### diagramUri

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>The URI of the architecture resource to generate an image for.</td>
</tr>
</tbody>
</table>

### getTiledImage ()

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITypedImage</td>
<td>Called when the web UI requests a specific tile in a large image. Parameters are diagramUri and tileId are passed in and used to determine exactly which part of the image is being requested. The implementation of this method is expected to return the typed image of the tile in one of the requested formats. The tileId MUST be specified in the tiled image descriptor object returned from a previous call to the getTile() method.</td>
</tr>
</tbody>
</table>

### jfsClient

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IJfsClient</td>
<td>An authenticated reference to a IJfsClient object that implementations can use to invoke SII and JFS services.</td>
</tr>
</tbody>
</table>

### diagramUri

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>The URI of the architecture resource to generate an image for.</td>
</tr>
</tbody>
</table>

### tileId

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>A unique id (relative to the diagramUri) that indicates the tile requested.</td>
</tr>
</tbody>
</table>

### acceptableFormats

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String[]</td>
<td>Array of acceptable mime types (image/png, image/jpg) ordered by preferred.</td>
</tr>
</tbody>
</table>
6.5.3 Repository Management

Repository management is a layered architecture in which the top layer serves some application in the server such as a mediation component, an agent which works against some external tool, or an internal application which performs some value-added function. The common to all these components is that they use RDF model data in the RDF store.

In the next Figure 25, that common layer used by all these components is named the SII-API layer. In the distributed configuration, there are multiple storage components, one of which is the Local RDF. The other servers (each of which maintains its own local RDF), are represented in this figure as the Remote RDF relative to the local).

The remote server is represented by two components: the Friend Server and the Remote RDF. The other components are all part of the local server where the application resides and where the distributed storage operation is initiated. The pair of components Friend Client and Friend Server interact using OSLC API under the security wrapping of the OAuth protocol [25] since they operate over the (hostile) internet.

![Figure 25 – Distributed DANSE tools-net server sequence diagram](image)

The SII-SPI component in this diagram spawns two parallel activities (which may also be in sequence w/out loss of generality) to perform the operation on both the local and remote RDF stores. The diagram also includes the possibility of an error resulting from any of the storage managers.

The details of this interaction are as follows:

- 1. Query – the operation is initiated.
  - 1.1. localQuery is issued against the local RDF store.
    - 1.1.1. Local RDF may fail the query which may affect the final results from the SII-API but not necessarily a complete failure.
    - 1.1.2. Alternatively, the Local RDF succeeds in serving the query and returns with query results.
  - 1.2. remoteQuery is issued in parallel to the localQuery and goes to the Friend Client component.
    - 1.2.1. Local Friend issues a friendQuery into the remote SII server, serviced via its Friend Server component
      - 1.2.1.1. Friend Server performs OAuth authentication for the legality of this access.

Version | Status  | Date       | Page
---|---------|------------|---
1.1 | Final   | 2012-06-04 | 45 of 56
1.2.1.2. The result of the OAuth may be an error in which case the remote query will not continue.

1.2.1.2.1. Friend Client forwards the failure from the Friend Server to the initiating SII-API.

1.2.1.3. Friend Server forwards the authenticated query to the Remote RDF store to perform the service requested.

1.2.1.3.1. A reply for the query is generated back to the issuing Friend Server.

1.2.1.3.1.1. The reply is forwarded again to the Friend Client as a successful remote operation result.

1.2.1.3.1.1.1. Friend Client returns the result as a remoteReply to the SII-API.

1.2.1.3.1.1.1.1. The SII-API, having both results from the remote and local storage managers applies a federation merging procedure which will accommodate the generally three possible results: all successful, some failures but not all, and a total failure.

1.2.1.3.1.1.1.1.1. Reply to the application is the result of the federated procedure consisting of one of three possible results: failure, success, or partial success.

End of description.

6.6 Support for multiple meta-models

Adopting the semantic web technology relies on applying semantic description of model data. Every modelling resource from tools-net tools should be tagged with the proper type which is defined in some ontology. That will allow to process model data, apply rules that imply relations among such elements and improve the general usefulness of the model data for the holistic view of the SoS over all tools, components and applications used to engineer its life-cycle.

The ontology standard adopted in DANSE (see 4.1 “Foundation standards of the tool-net”) is OWL 2.0 [26] for which there are several options to generate. Firstly, some ontologies on which tools ontologies rely on (e.g. import) or use can be accessed over the internet. Ontologies can be edited with tools such as Protégé [36], or over Jazz/DM.

An ontology for tools models must use an ontology which sufficiently describe the models exported from and imported into these tools so that these activities can operate under semantic mediation rules in an automatic way. When using the DM domain modelling in which OWL 2.0 ontologies can be defined, there is an added value to them since the models for these domains can be managed within DM and be shared among tool instances all of which can work with the same modelling language. This means, for instance, several Rhapsody tools, or tools which can work with the same internal format (or via the published API) of the Rhapsody product.

Some of the models in DANSE, those serving the common semantics or reference models for the different tools working in the different modelling domains identified in DANSE (see for instance 5.3 “Modelling and tools domains relevant for DANSE”) – should be defined using Jazz DM domain definitions.

6.6.1 Using jazz/DM meta-modelling

The development of services within DM is done with the DM SDK – Software Development Kit. This kit consists of Java jars which support the entire front-end services composing the DM component, and the configuration needed to execute that as a separate server working in parallel to the JTS Jazz server.

Developing a DM service is very much like developing any web service, adhering to the bit different interface defined by the DM framework. The DM framework is itself a web service wrapping such “sub” services. The wrapper provides user authentication and authorization services, which are implemented through the OAuth protocol.

The semantic mediation for DANSE is such “sub” service hosted on the DM framework. This is implemented as another layered framework in which semantic mediation services (modules) are embedded and activated.
by that framework implementation. Being a DM service, it will also provide the graphics user interface by which the mediation can be designed, configured and monitored.

DTK is a web tool-kit that DM implements as a service under its umbrella. With this tool user can author OWL ontology for a tool or intermediary modeling stage artifact, or a final modeling artifact in the SSI. For instance here is a DTK domain modeling screen of a BSO model is shown in Figure 26.

![Figure 26: Authoring ontology with the DTK](image)

### 6.7 Domain interoperability implementation

#### 6.7.1 Preface

The following is a contribution by partner SODIUS to address the problem of interoperability of multiple modeling tools and is an alternative to the Jazz/DM solution described above. In general, the use of OSLC is a foundation feature of the tool-net. Yet, the approach introduced above is to do interoperability over the DANSE platform and not via direct tool-to-tool interoperability. To reach that end, DANSE tool-net will use semantic mediation methods as explained above. However, as seen below – there are more than one ways to “skin the cat” and that will be discussed among the technical partners to find best practices to be used.

#### 6.7.2 Tools interoperability (SODIUS)

Multiple authoring tools or open source platforms are used to support the different data models in the tool-net platform. For the moment, integration provided between those tools is not well-defined and this leads to a fragile interoperability development environment. Tool-Net aims to reify and support as domain-specific domains those data-model and capability to link them in a distributed environment using OSLC.
From OSLC, the tool-net platform will benefit from open specifications that enable tools from different domains to provide common integration scenarios, guidelines and technical assets to initiate user interface, client and provider implementation.

Several technical points will have to be addressed for each connected tool:
- Define at least client (and for some of tools providers) to connect applications,
- Define transformation rules between semantic domains,
- Automate and deploy those rules.

This could be done for each tool in a specific way. But considering we aim to address several scenarios another way to deal with this task is to define a more productive and perhaps simplest way to achieve those integrations. One possibility is to use existing model-driven platform as MDWorkbench (SODIUS) to benefit from raw data importer/exporter and native model-driven transformation capabilities.

Implementing a kind of model “hub”, MDWorkbench is a model-based interoperability platform that already defines generic connector capabilities (Rhapsody and UML2 modellers, DOORS, MATLAB Simulink, System Architect, MEGA). By adding OSLC support to the generic MDWorkbench platform, it would be possible to define in a quicker, productive and unified way several OSLC clients and providers for the tool-net platform, acting as an “OSLC wrapper” factory.

MDWorkbench server extension would act as a “rules” service provider between different domains. Considering the need of transformation rules, MDWorkbench runtime assets could be in this way deployed to handle complex transformation processing (in opposition with more simple link creation scenarios where light way integration is sufficient).

![Diagram of MDWorkbench interfacing models for the tools-net](image)

**Figure 27:** MDWorkbench interfacing models for the tools-net
7 Execution plans

The tools-net workplan of WP8 is as follows:

![WP8 workplan diagram]

The T8.1 task ends with the delivery of this document and in parallel, the work on T8.3 for the tools-net architecture continues with an interest to have presentable results in the M12 review – which are yet to be defined. A first prototype is planned only by M18.

Toward that prototype, the Jazz platform which will support the design tools and applications of DANSE will be populated with adapted tools as well as be prepared to be shared by all partners as an SDK – along the outlines in the architecture chapter.

Suggestions by partners, specifically SODIUS, for tools and services that can work on the Jazz platform - as well as tool enhancements that can work with that platform – will be an ongoing work item in the following months leading towards the first prototype as well as the early show case of the Y1 review.

7.1 The prototypes

DANSE plans 3 prototypes on the M18, M24 and M30 milestones. At this point in time (M6), DANSE partners are starting to deal with the contents that can be demonstrated in the first year review in M12, in what is termed a “showcase”. At the time of writing these lines, that part is still forming up.

The content of the first prototype are not yet defined. However, it is certain that the following will be part of this prototype:

1. A Jazz/DM platform that can serve partners using some of the tools in Table 2 for which there is a decision to support within the DANSE framework.
2. The platform will have an initial support for model mediation.
3. An effort will be made to create several ontologies for common models to serve some of the domains of DANSE. These ontologies will serve to share models among the tools selected for this milestone.
4. Applications on the Jazz/DM platform will provide management over the collaborative DANSE design process, as well as possible value-added functions – TBD.
5. A distributed instance of the DANSE tool-net over the internet using more than one yellow-zone machines will be installed. One of the partners installing a DANSE server is IBM Haifa, while at least one additional partners still needs to be identified.

Any further assumptions about this prototype are too early to predict at this point as the particular use cases to be tried on it are not yet defined.
8 Summary

The DANSE tools-net concepts and architecture have been described with references to the different work packages, previous relevant projects, and the requirements.

Both this document as well as the requirements document has been formed in parallel, although there is a clear dependency of this document on the requirements. In fact the requirements document D3.1 has not been turned in yet and it still at its final stages. At the same time, the D3.1 document is considered a “live” document to be continuously modified as the understanding of the DANSE problem area progresses.

For that reason, only the initial users’ requirements have been directly dealt with in this document. The road to the first prototype, leading through the earlier review of DANSE year 1 will complement this document with more concrete concepts and architecture as well as we expect that the set of DANSE tools will be closing up as well.

8.1 Thanks

Thanks to WP leaders and participants who provided background material for the respective chapter of this document, and the DANSE members who participated in the review sessions.
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.

**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.

### GIS
Geographic Information System

### 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.

### 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 (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.

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
10 References

[10] UPDM  "Unified Profile for DoDAF and MoDAF"
[12] MoDAF  
[13] NAF  

Version  | Status  | Date      | Page
--- | --- | --- | ---
1.1  | Final  | 2012-06-04  | 55 of 56
[25] OAuth

[26] OWL

[27] Linked Data

[28] SysML

[29] Esri

[30] RTP
Christian Hein and Tom Ritter, “Lessons Learned Implementing the CESAR Reference Technology Platform,”
http://www.cesarproject.eu/fileadmin/user_upload/ATC/1_4_ATC_CESAR_RTP.pdf

[31] D3.1
D3.1 " Project Requirements"

[32] Alexander’77

[33] OWA

[34] RESTful
"Representational State Transfer (REST),"
http://en.wikipedia.org/wiki/Representational_State_Transfer

[35] ECSAM

[36] Protégé