ECSEL Research and Innovation actions (RIA)

AMASS

Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems

Methodological Guide for Seamless Interoperability (b) D5.8

Work Package:	WP5: Seamless Interoperability
Dissemination level:	PU =Public
Status:	Final
Date:	31st October 2018
Responsible partner:	Tomáš Kratochvíla (Honeywell)
Contact information:	Tomas.Kratochvila@Honeywell.com
Document reference:	AMASS_D5.8_WP5_HON_V1.0

This deliverable is part of a project that has received funding from the ECSEL JU under grant agreement No 692474. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and from Spain, Czech Republic, Germany, Sweden, Italy, United Kingdom and France.

Contributors

Names	Organisation
Tomáš Kratochvíla, Vít Koksa	Honeywell (HON)
Jose Luis de la Vara, Jose María Álvarez, Francisco Rodríguez, Eugenio Parra, Fabio di Ninno, Miguel Rozalen	Universidad Carlos III de Madrid (UC3)
Luis M. Alonso, Borja López, Julio Encinas	The REUSE Company (TRC)
Pietro Braghieri, Stefano Tonetta, Alberto Debiasi	Fondazione Bruno Kessler (FBK)
Morayo Adedjouma, Botella Bernard, Huascar Espinoza, Thibaud Antignac	CEA LIST (CEA)
Marc Sango	ALL4TEC (A4T)
Ángel López, Alejandra Ruiz	Tecnalia Research and Innovation (TEC)
Jan Mauersberger	medini Technologies AG (KMT)

Reviewers

Names	Organisation
Eugenio Parra (Peer reviewer)	Universidad Carlos III de Madrid (UC3)
Jaroslav Bendík (Peer reviewer)	Masaryk University (UOM)
Cristina Martinez (Quality Manager)	Tecnalia Research and Innovation (TEC)
Alejandra Ruiz Lopez (TC reviewer)	Tecnalia Research and Innovation (TEC)

TABLE OF CONTENTS

List of Figures

List of Tables

List of Listings

Executive Summary

The deliverable *D5.8 Methodological Guide for Seamless Interoperability (b)* is released by the AMASS work package *WP5 Seamless Interoperability* and provides information about how to use the approaches and tools for seamless integration of engineering tools. This deliverable is the second outcome of the task T5.4 "Methodological Guidance for Seamless Interoperability" and is based on the results from tasks T5.1 "Consolidation of Current Approaches for Seamless Interoperability" (*D5.1 Baseline requirements for seamless interoperability* [10]), the outputs of the task T5.2 "Conceptual Approach for Seamless Interoperability" (*D5.2 Design of the AMASS tools and methods for seamless interoperability (a)*, *D5.3 Design of the AMASS tools and methods for seamless interoperability (b)* [11]), and on the three outputs of the task T5.3 Implementation for Seamless Interoperability (*D5.4 Prototype for seamless interoperability (a)* [12], *D5.5 Prototype for seamless interoperability (b)* [13], and *D5.6 Prototype for seamless interoperability (c)* [14]).

The guide contains a set of rules to use the architecture and usage scenarios with detailed steps. The intention is that third parties such as tool vendors can apply these guidelines to connect their tools to other tools in the scope of seamless interoperability.

Anyone should be able to integrate their tools with the AMASS Platform using this guide. Moreover, this guidance is applicable for seamless tool integration in general, for example for the following integration frameworks: OSLC (KM, Automation), Papyrus, or ad-hoc integration.

This document focuses on the guidelines for the techniques developed in WP5 for Seamless Interoperability. To have more general overview and guidelines for the AMASS approach including the methods and techniques provided by other WPs, the reader is referred to D2.5 (AMASS user guidance and methodological framework) [9]. In particular, the WP5 activities can be enriched with the link to reference standards.

The main relationships of D5.8 with other AMASS deliverables are as follows:

- D2.1 (Business cases and high-level requirements) [8] includes the requirements that the design for Seamless Interoperability must satisfy.
- D2.4 (AMASS reference architecture (c)) [19] presents the high-level architecture of the AMASS Tool Platform.
- D5.1 (Baseline requirements for seamless interoperability) [10] reviews and consolidates existing work for Seamless Interoperability.
- D5.3 (Design of the AMASS tools and methods for seamless interoperability (b)) [11] is the final version of the AMASS design for Seamless Interoperability.
- D5.6 (Prototype for seamless interoperability (c)) [13] reports how the design in D5.3 has been implemented in the AMASS Prototype P2.

1. Introduction

Embedded systems have significantly increased in technical complexity towards open, interconnected systems. The rise of complex Cyber-Physical Systems (CPS) has led to many initiatives to promote reuse and automation of labour-intensive activities such as the assurance of their dependability. The AMASS approach focuses on the development and consolidation of an open and holistic assurance and certification framework for CPS, which constitutes the evolution of the OPENCOSS [15] and SafeCer [16] approaches towards an architecture-driven, multi-concern assurance, reuse-oriented, and seamlessly interoperable tool platform.

The expected tangible AMASS results are:

a) The **AMASS Reference Tool Architecture**, which will extend the OPENCOSS and SafeCer conceptual, modelling and methodological frameworks for architecture-driven and multi-concern assurance, as well as for further cross-domain and intra-domain reuse capabilities and seamless interoperability mechanisms (based on OSLC specifications [17]).

b) The **AMASS Open Tool Platform**, which will correspond to a collaborative tool environment supporting CPS assurance and certification. This platform represents a concrete implementation of the AMASS Reference Tool Architecture, with a capability for evolution and adaptation, which will be released as an open technological solution by the AMASS project. AMASS openness is based on both standard OSLC APIs with external tools (e.g. engineering tools including V&V tools) and on open-source release of the AMASS building blocks.

c) The **Open AMASS Community**, which will manage the project outcomes, for maintenance, evolution and industrialization. The Open Community will be supported by a governance board, and by rules, policies, and quality models. This includes support for AMASS base tools (tool infrastructure for database and access management, among others) and extension tools (enriching the AMASS functionality). As Eclipse Foundation is part of the AMASS consortium, the Polarsys/Eclipse community (www.polarsys.org) has been selected to host AMASS Open Tool Platform.

To achieve the AMASS results, as depicted in Figure 1, the multiple challenges and corresponding scientific and technical project objectives are addressed by different work-packages.

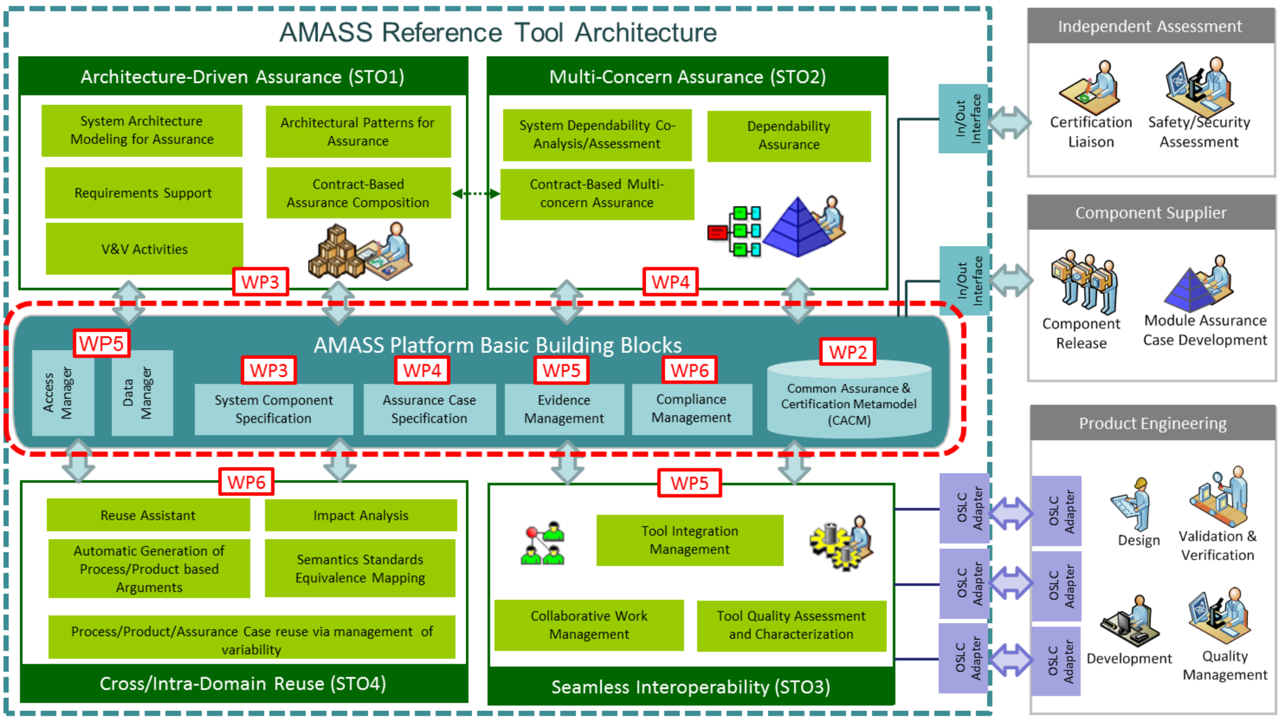

AMASS Reference Tool Architecture
Architecture-Driven Assurance (STO1)
System Architecture Modeling for Assurance
Architectural Patterns for Assurance
Requirements Support
Contract-Based Assurance Composition
V&V Activities
WP3
Multi-Concern Assurance (STO2)
System Dependability Co-Analysis/Assessment
Dependability Assurance
Contract-Based Multi-concern Assurance
WP4
In/Out Interface
Independent Assessment
Certification Liaison
Safety/Security Assessment
Component Supplier
Component Release
Module Assurance Case Development
AMASS Platform Basic Building Blocks
WP5
Access Manager
Data Manager
WP3
System Component Specification
WP4
Assurance Case Specification
WP5
Evidence Management
WP6
Compliance Management
WP2
Common Assurance & Certification Metamodel (CACM)
Product Engineering
Design
Validation & Verification
Development
Quality Management
OSLC Adapter
WP6
Reuse Assistant
Impact Analysis
Automatic Generation of Process/Product based Arguments
Semantics Standards Equivalence Mapping
Process/Product/Assurance Case reuse via management of variability
Cross/Intra-Domain Reuse (STO4)
WP5
Tool Integration Management
Collaborative Work Management
Tool Quality Assessment and Characterization
Seamless Interoperability (STO3)

Figure 1. AMASS Prototype P2 building blocks

Since AMASS targets high-risk objectives, the AMASS Consortium decided to follow an incremental approach by developing rapid and early prototypes. The benefits of following a prototyping approach are:

- Better assessment of ideas by initially focusing on a few aspects of the solution.
- Ability to change critical decisions based on practical and industrial feedback (case studies).

AMASS has planned three prototype iterations:

1. During the **first prototyping** iteration (Prototype Core), the AMASS Platform Basic Building Blocks (see Figure 1), will be aligned, merged and consolidated at TRL4[1].
2. During the **second prototyping** iteration (Prototype P1), the AMASS-specific Building Blocks will be developed and benchmarked at TRL4; this comprises the blue basic building blocks as well as the green building blocks (Figure 1). Regarding seamless interoperability, in this second prototype, the specific building blocks will provide advanced functionalities regarding tool integration, collaborative work, and tool quality characterisation and assessment.
3. Finally, at the **third prototyping** iteration (Prototype P2), all AMASS building blocks will be integrated in a comprehensive toolset operating at TRL5. Functionalities specific for seamless interoperability developed for the second prototype will be enhanced and integrated with functionalities from other technical work packages.

Each of these iterations has the following three prototyping dimensions:

- ***Conceptual/research development***: development of solutions from a conceptual perspective.
- ***Tool development***: development of tools implementing conceptual solutions.
- ***Case study development***: development of industrial case studies (see D1.1 [18]) using the tool-supported solutions.

[1] In the context of AMASS, the EU H2020 definition of TRL is used, see http://ec.europa.eu/research/participants/data/ref/h2020/other/wp/2016_2017/annexes/h2020-wp1617-annex-g-trl_en.pdf

As part of the Prototype Core, WP5 was responsible for consolidating the previous works on specification of evidence characteristics, handling of evidence evolution, and specification of evidence-related information (e.g. process information) in order to design and implement the basic building block called "Evidence Management" (see Figure 1). In addition, WP5 was responsible for the implementation of the "Access Manager" and "Data Manager" basic building blocks. Nonetheless, the functionality of these latter blocks is used not only in WP5, but in all the WPs, e.g. for data storage and access (of system components, of assurance cases, of standards' representations, etc.). For P1 and P2 prototypes, WP5 has refined and extended the existing implementation with support for specific seamless interoperability based on the development of new functionality, and not only the integration of available tools.

This deliverable is the output of Task T5.4 "Methodological Guidance for Seamless Interoperability".

It is a methodological guide to use the Seamless Interoperability approach. The guide contains a set of rules to use the architecture and usage scenarios with detailed steps. The intention is that 3rd-parties like tool vendors can apply these guidelines to connect their tools to other tools in the scope of seamless interoperability.

There are several possible approaches to establish an effective interconnection of various systems. The approaches most relevant to the AMASS platform are discussed at a conceptual level in the Chapter 2.

Chapter 3 contains more detailed description of the individual approaches and some practical hints related to their implementation.

Appendix A recommends the procedure of selecting an appropriate solution for a given integration task. The procedure is captured in the form of EPF (Eclipse Process Framework) process description.

Appendix B contains lots of practical information about the OSLC KM approach.

Appendix C contains the list of modifications with respect to the predecessor of this document, i.e. to the D5.7 Methodological Guide for Seamless Interoperability (a).

2. Seamless Interoperability Approaches

This section presents the technology-specific interoperability approaches that are supported and implemented for seamless interoperability of 3rd party tools in AMASS.

2.1 Evidence Management

Assurance evidence corresponds to artefacts that contribute to developing confidence in the dependable operation of a system and that can be used to show the fulfilment of the criteria of an assurance standard. Examples of artefact types that can be used as assurance evidence include risk analysis results, system specifications, reviews, testing results, and source code. Those artefacts that correspond to assurance evidence can be referred to as *evidence artefacts*. The *body of assurance evidence* of an assurance project is the collection of evidence artefacts managed. A *chain of assurance evidence* is a set of pieces of assurance evidence that are related, e.g. a requirement and the test cases that validate the requirement. *Assurance evidence traceability* is the degree to which a relationship can be established to and from evidence artefacts. *Impact analysis of assurance evidence change* is the activity concerned with identifying the potential consequences of a change in the body of assurance evidence.

Evidence management can be defined as *the system assurance and certification area concerned with the collection and handling of the body of assurance evidence of an assurance project*. When managing assurance evidence, the first step is usually to determine what evidence must be provided. Afterwards, the evidence artefacts must be collected and might also have to be evaluated and traced to other artefacts. During this process, it might be necessary to make changes in the evidence artefacts, and such changes might impact other items. Once the body of evidence of the assurance project is regarded as adequate, the process can be finished.

2.2 OSLC KM

In this section, the main building blocks of the OSLC KM specification are outlined. The OSLC KM motivation, objectives and shape have been already introduced in the Deliverable D5.3 [11] (section 3.1.1). Here, we recall the main concepts of OSLC and the Knowledge Management Specification:

- The Open Services for Lifecycle Collaboration (OSLC) initiative is a joint effort between academia and industry to boost data sharing and interoperability among applications by applying the Linked Data principles: "*1) Use URIs as names for things. 2) Use HTTP URIs so that people can look up those names. 3) When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL) and 4) Include links to other URIs, so that they can discover more things*".
- OSLC is based on a set of specifications that take advantage of web-based standards such as the Resource Description Framework (RDF) and the Hypertext Transfer Protocol (HTTP) to share data under a common data model (RDF) and protocol (HTTP). Every OSLC specification defines a *shape* for a particular type of resource. For instance, requirements, changes, test cases or estimation and measurement metrics, to name a few, have already a defined shape (also called OSLC Resource Shape). In this context, last times have also seen the creation of two new approaches for defining data shapes: the Shape Expressions[2] (ShEx) language to describe RDF graph structures and the Shapes Constraint Language[3] (SHACL), a W3C Recommendation. In both cases, these languages allow developers to define the structure of the data to be exchanged with the aim of validating RDF documents, communicating expected graph patterns for potential reuse in APIs and to generate

[2] https://www.w3.org/2001/sw/wiki/ShEx

[3] https://www.w3.org/TR/shacl/

user interface forms and code. In the case of OSLC, the resource shape serves us to define the structure of the data to be exchanged. There is a common core of properties and elements, the OSCL Core vocabulary, and, then, depending on the domain there are extensions to the core vocabulary creating a domain specific language for each artefact type to be exchanged.

Although OSLC Resource Shapes or, more precisely, data shapes, have already been defined to model metadata and contents of different types of artefacts, there are still some types of artefacts for which there is no shape. As examples, it is possible to find elements of a vocabulary, an ontology, an electrical circuit, a requirements pattern or a dynamic system model to name a few. Due to this situation, a common strategy for knowledge management is hard to draw since no common representation language for any kind of artefact is available (here RDF is used as underlying data model but a vocabulary on top of that is completely necessary). Therefore, a universal data shape, as presented in Deliverable D5.3 [11] (section 3.1.1), is required. The System Representation Language (SRL) is the vocabulary designed to represent information for any type of artefact generated during the development lifecycle. This data shape gives a response to accommodate the processes in a knowledge management strategy, see Table 1:

Table 1. Mapping of the OSLC KM approach to the knowledge management processes

Knowledge Management Process	Support
Capture/Acquire	Access OSLC repositories in the context of Systems Engineering for all existing specifications and other RDF-based services or SPARQL endpoints.
Organize/Store	RDF as a public exchange data model and syntax, and as a universal internal representation model to build the System and Software Knowledge Repository (SKR).
Access/Search/Disseminate	RDF query language (e.g. SPARQL), natural language or a native query language (if any). A set of entities and relationships creating an underlying graph.
Use/Discover/Trace/Exploit	Entity reconciliation based on graph comparison.
Visualization	A generic graph-based visualization framework that can show not only entities and relationships, but also interpret them as type of diagram. E.g. Class diagram.
Exploit	Index, search, trace or assess quality based on the internal representation model.
Share/Learn	An OSLC interface on top of the SKR that offers both data and services.
Create	Third-party tool that exports artefacts using an OSLC-based interface.

On the other hand, and due to the fact that a huge amount of data, services and endpoints based on RDF and the Linked Data principles are already publicly available, a mapping between any RDF vocabulary and the data shape is completely necessary to support backward compatibility and to be able to import any piece of RDF data into an OSLC KM based repository.

Building on these assumptions and considering the guidelines and definitions of the OSLC Core specification, the data shape for knowledge management (the SRL vocabulary) will conform the next basic OSLC definitions:

1. "*An OSLC Domain is one ALM (Application Lifecycle Management) or PLM (Product Lifecycle Management) topic area*". Each domain defines a specification.

   In this case, a new domain is being defined: Knowledge Management (KM).

2. *"An OSLC Specification is comprised of a fixed set of OSLC Defined Resources*".

   According to the Deliverable D5.3 [11], the SRL vocabulary will be used as the underlying shape for knowledge items. The key concepts of this metamodel are the Artefact and Relationship classes. An artefact is a container of relationships (Relationships) that can have metaproperties being that metadata (e.g. authoring, versioning, visualization features and, in general, provenance information) and artefact properties (e.g. maxTolerance, refTemperature, etc.). An artefact can also own other sub artefacts to support situations such as "a model has different diagrams". If an artefact only represents the apparition of a term it will contain a reference to a term (element of a controlled vocabulary or taxonomy). This term can have a grammatical category (TermTag) such as name, pronoun, adverb or verb to name just a few. In the same manner, a semantic category (SemanticCluster) represented by a term can be assigned to a term for instance the semantics "negative". Thus, different terms can have different semantics. Finally, a relationship establishes a link between *n* artefacts and semantics that can be also attached to the link, e.g. "part-of" (by default a relationship will be considered as a composition). Figure 2 presents an updated version of the SRL elements.

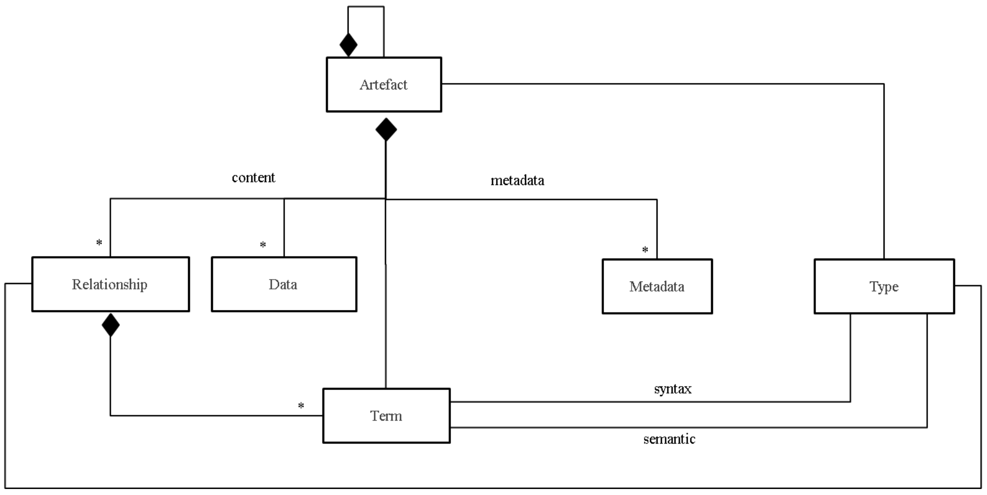


Figure 2. Updated UML Class Diagram of the OSLC Knowledge Management Resource Shape

3. *"An OSLC Defined Resource is an entity that is translated into an RDF class with a type".* Every resource consists of a fixed set of defined properties whose values may be set when the resource is created or updated.

   Considering that the Linked Data Initiative has seen in recent times the creation of methodologies, guidelines or recipes to publish RDF-encoded data, we have paid special attention to follow a similar approach by reusing existing RDF-based vocabularies. More specifically, the following rules have been applied to create the OSLC resource shapes:

   - If there is an RDF-based vocabulary that is already a W3C recommendation or it is being promoted by other standards organization, it must be used as it is, by creating an OSCL Resource Shape.
   - If there is an RDF-based vocabulary but it is just a *de-facto* standard, it should be used as it is, by including minor changes in the creation of an OSCL Resource Shape.
   - If there is not an RDF-based vocabulary, try to take advantage (reusing properties and classes) of existing RDF-based vocabularies to create the OSLC Resource Shape.

   In the particular case of knowledge management, we have selected the Simple Knowledge Organization System (SKOS), a W3C recommendation, to define concepts, since it has been designed for promoting controlled vocabularies, thesauri, taxonomies or even simple ontologies to the Linked Data initiative. That is why, in our model, most of the entities can be considered as a *skos:Concept* and we have created the shape of this standard definition of concept in the resource *oslc_km:Concept*.

4. *"An OSLC Defined Property is an entity that is translated into an RDF property"*. It may define useful information such as the type of the property, datatypes and values, domain, range, minimum and maximum cardinality, representation (inline or reference) and readability.

   The detailed description of all properties for every defined resource is an evolution (and extension) of the initial shape defined in the public deliverable "Interoperability Specification - V2" of the CRYSTAL project[4], a summary of these defined properties is also presented in *Appendix B. The OSLC KM Resource Shape*.

5. *An OSLC Service Provider is a tool that offers data implementing an OSLC specification in a REST-fashion.*

   The Figure 3 shows a functional architecture for an OSLC Knowledge Management provider. It shall be able to process any kind of OSLC-based resource or even any piece of RDF. Once the data is in the OSLC-KM processor, a reasoning process can be launched to infer new RDF triples (if required). Afterwards, data is validated and indexed into the system and software knowledge repository (SKR). On top of this repository, services such as semantic search, naming, traceability, quality checking or visualization may be provided, generating new OSLC KM Resources. This functional architecture has a reference implementation on top of Knowledge Manager [20].

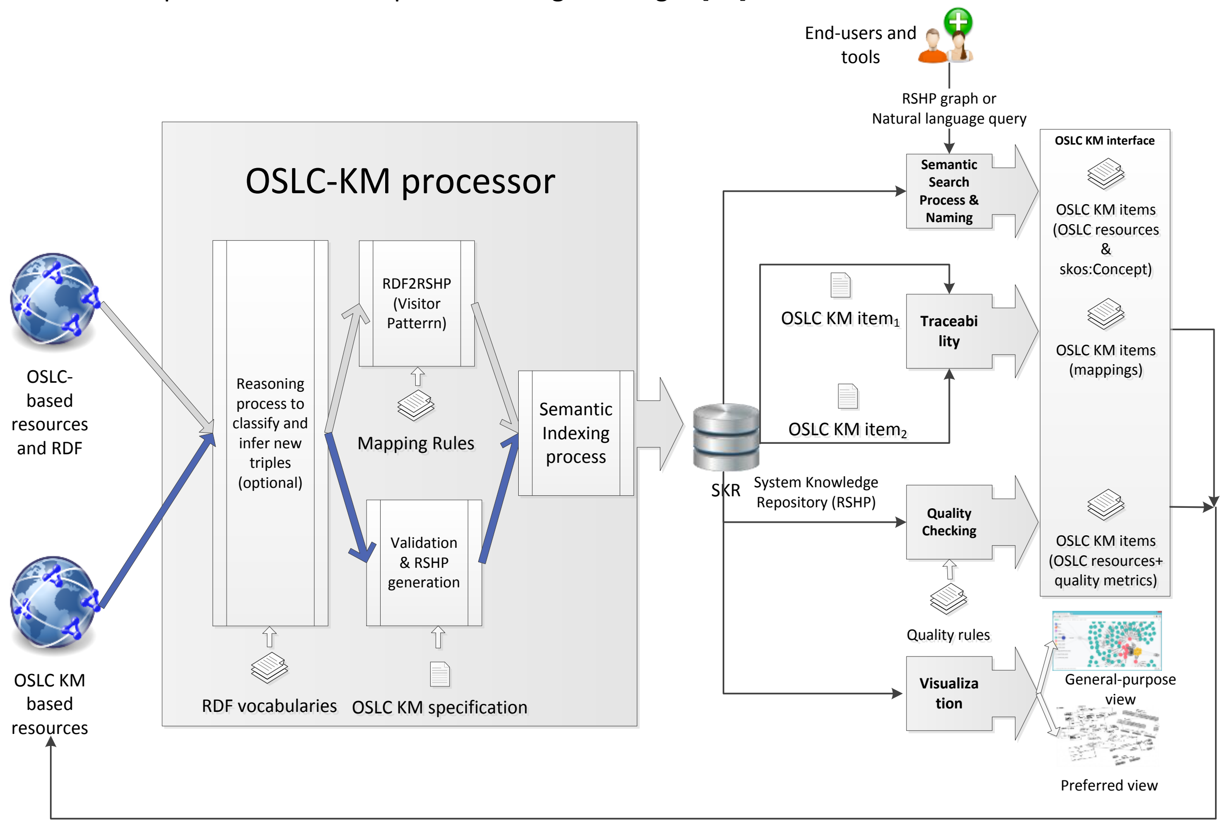

End-users and tools
RSHP graph or Natural language query
OSLC-KM processor
OSLC-based resources and RDF
OSLC KM based resources
Reasoning process to classify and infer new triples (optional)
RDF2RSHP (Visitor Patterrn)
Mapping Rules
Validation & RSHP generation
Semantic Indexing process
RDF vocabularies
OSLC KM specification
SKR
System Knowledge Repository (RSHP)
OSLC KM item1
OSLC KM item2
Semantic Search Process & Naming
Traceability
Quality Checking
Quality rules
Visualization
OSLC KM interface
OSLC KM items (OSLC resources & skos:Concept)
OSLC KM items (mappings)
OSLC KM items (OSLC resources+ quality metrics)
General-purpose view
Preferred view

Figure 3. Functional Architecture and core services for knowledge management based on the OSLC KM

2.2.1 Mapping Between Any Piece of RDF to the OSLC KM Data Shape

The emerging use of RDF to tackle interoperability issues in different contexts has created a data-based environment in which data and information can be easily exchanged. Given this situation a strategy to map

[4] The deliverable can be found in the next URL: http://www.crystal-artemis.eu/fileadmin/user_upload/Deliverables/CRYSTAL_D_601_023_v3.0.pdf and an up-to-date version is available in http://trc-research.github.io/spec/km/ (Last access: October 2018).

RDF-encoded data to the OSLC KM Resource Shape (hereafter KM) abound must be defined similar to the one presented in the mapping between relational databases and RDF.

To do so a direct mapping is defined to perform simple transformations and to provide a basis for defining and comparing more complex transformations. In order to design this direct mapping, both models are represented using the commonly defined abstract data types set and list. This algebraic formalization of the core fragment of RDF to be translated into KM, that is, RDF without RDFS vocabulary and literal rules allows us to make a graph syntax transformation. The definitions follow a type-as-specification approach; thus models are based on dependent types that can also include cardinality. More specifically, Listing 1 shows both specifications as a kind of regular tree grammars that can be used to specify a rule-based transformation between two grammars (denotational semantics). Thus, a transformation between RDF and KM can be defined as a function, $\mathrm{RDF2KM}$, that takes the RDF grammar, $\mathrm{G_{RDF}}$, a valid RDF graph, $\mathrm{RDF_{graph}}$, the KM grammar $\mathrm{G_{km}}$ and a set of direct mapping rules, $\mathrm{M_{rdf2km}}$ (see Listing 2 where sub-indexes refer to attributes and relationships of the elements), to generate a valid $\mathrm{KM_{graph}}$.

$$RDF2KM\colon G_{RDF} \times RDF_{graph} \times G_{km} \times M_{rdf2km} \rightarrow Relationship_{graph}$$

(1) RDF Graph ::= Set(Triple) (2) Triple ::= (Subject, Predicate, Object) (3) Subject ::= IRI | BlankNode (4) Predicate ::= IRI (5) Object ::= IRI | BlankNode | Literal (6) BlankNode ::=RDF blank node (7) Literal ::=PlainLiteral | TypedLiteral (8) PlainLiteral ::= lexicalForm | (lexicalForm, languageTag) (9) TypedLiteral ::= (lexicalForm, IRI) (10) IRI ::= RDF URI-reference as subsequently restricted by SPARQL (11) lexicalForm ::= a Unicode String	(1) Artefact ::= (Set(Relationship), MetaData{0,*}) | (Term {0,1}) | (2) Relationship ::= (Subject, Verb, Predicate, Semantics) (3) Subject ::= Artefact {0,1} (4) Verb ::= Artefact {0,1} (5) Object ::= Artefact {0,1} (6) Term ::= (lexicalForm, languageTag, TermTag) (7) Type ::= lexicalForm (8) MetaData ::= (Tag, Value) (9) Term ::= { Artefact, lexicalForm} (10) Term ::= { Artefact {0,1}, lexicalForm {0,1}} (11) Term ::= (lexicalForm, languageTag, TermTag)

Listing 1. Regular Tree Grammars of RDF, $\boldsymbol{G_{RDF}}$, and the OSLC KM Resource Shape, $\boldsymbol{G_{km}}$

```
1. RDF Graph    ::= Artefact
2. Triple ::= Realtionship
3. Subject ::= ARTEFACT / ARTEFACT_Id =IRI, ARTEFACT_TERM=(lexicalForm=label(IRI),languageTag=”EN”,SyntaxTag=
   Realtionship.POS_TAGGING.CATEGORY)
4. Predicate ::= ARTEFACT / ARTEFACT_Id =IRI, ARTEFACT_TERM=(lexicalForm=label(IRI),languageTag=”EN”,SyntaxTag=
   Realtionship.POS_TAGGING.CATEGORY)
5. Object ::=
   5.1. ARTEFACT / ARTEFACT_Id =IRI, ARTEFACT_TERM=(lexicalForm=label(IRI),languageTag=”EN”,
        SyntaxTag= Realtionship.POS_TAGGING.CATEGORY) /*When the object is a resource.*/
   5.2. ARTEFACT / ARTEFACT_Id = auto_generate_id, ARTEFACT_TERM=(lexicalForm= PlainLiteral.
        lexicalForm,languageTag=PlainLiteral.languageTag,SyntaxTag= Realtionship.POS_TAGGING.CATEGORY)
        /*When the object is a PlainLiteral.*/
   5.3. ARTEFACT / ARTEFACT_Id = auto_generate_id, ARTEFACT_TERM=(lexicalForm= TypedLiteral.
        lexicalForm,languageTag= Realtionship.POS_TAGGING.CATEGORY.LANG,SyntaxTag= TypedLiteral.IRI)
        /*When the object is a TypedLiteral.*/
6. BlankNode ::= ARTEFACT / ARTEFACT_Id =IRI,
   ARTEFACT_TERM=(lexicalForm=label(IRI),languageTag=”EN”,SyntaxTag=RDF.BLANK_NODE)
7. Literal ::=    PlainLiteral | TypedLiteral
8. PlainLiteral ::=
   8.1. Term / Term_lexicalForm = lexicalForm, Term_languageTag = Relationship.POS_TAGGING.LANG, Term_syntaxTag =
        Relationship.POS_TAGGING.CATEGORY |
   8.2. Term / Term_lexicalForm = lexicalForm, Term_languageTag = languageTag, Term_syntaxTag =
        Relationship.POS_TAGGING.CATEGORY
9. TypedLiteral ::= Term / Term_lexicalForm = lexicalForm, Term_languageTag = Relationship.POS_TAGGING.LANG,
   Term_syntaxTag = IRI
10. LexicalForm ::= Term_lexicalForm
11. IRI ::= lexicalForm=label(IRI)
```

Listing 2. Set of mapping rules, $\mathbf{M_{rdf2km}}$, to transform RDF in OSLC KM Resource Shape

2.2.2 Initial Assessment of the OSLC KM Data Shape

Taking into account the functional architecture presented in Figure 3 to exploit OSLC/RDF/SRL, a new question arises: *Which is the gain of having a common representation model?*

In order to address this question, it is necessary to establish a context in which data and information is being exchanged. Assuming that there is a common and shared data model (RDF) and a set of standard protocols to access this information (HTTP-based technology), we will focus on the gain of using the presented approach to provide a set of core and common services.

Let $\mathrm{s_k}$ be a service providing data and information, according to the OSLC principles. To represent the data exchanged in this service, a data shape comprising a set of RDF-based vocabularies, $\mathrm{V_{RDF}}$, is being used.

Let also $\mathrm{c_k}$ be a client of this service $\mathrm{s_k}$, if this client wants to automatically consume and process the data provided by $\mathrm{s_k}$ then it must necessarily process the set $\mathrm{V_{RDF}}$ so that, at least, $\#\mathrm{V_{RDF}}$ (cardinality of the set $\mathrm{V_{RDF}}$) mappings are necessary.

If we generalize this situation to an environment in which there is a set S of service providers publishing data under a set of RDF-based vocabularies where $\mathrm{V_{RDF}^{s_k}}$ represents the set of RDF-based vocabularies used by the service $\mathrm{s_k}$, a client of this set S must create $\sum \#\mathrm{V_{RDF}^{s_k}}$ mappings.

Furthermore, if we assume that we will likely have a set of clients C, and the set of mappings or adapters will not be publicly shared, we can easily infer that the total number of required mappings (in the worst case) to provide an interoperable environment raises to: $\#C \cdot \sum \#V_{RDF}^{s_k}$ which implies a large amount of time and effort for developing the same task.

On the other hand, the presented approach needs just one mapping, since there is a set of generic mapping rules between any RDF-based vocabulary and the KM shape. Thus, in order to provide a set of core services, see Figure 3, only one set of mapping rules is required, easing the task of consuming data exchanged under different RDF-based vocabularies, by providing not just a data model for this exchange, but a data model to universally represent any kind of information.

There is also another positive side-effect of applying the presented approach, if a service s_k wants to publish a certain data set then it must necessarily use a set of RDF-based vocabularies $V_{RDF}^{s_k}$ to represent such information.

2.3 V&V Manager and OSLC Automation

OSLC Automation specification builds on the Open Services for Lifecycle Collaboration (OSLC) Core Specification to define the OSLC resources and operations supported by an OSLC automation provider. The full specification is available at:
http://open-services.net/wiki/automation/OSLC-Automation-Specification-Version-2.1/

Optionally, the integration of FBK V&V Tools based on OSLC Automation could be used without the V&V Manager as described in section 2.6.

OSLC Automation allows not only automation of tasks and processes, but also an integration of any non-interactive tool without writing specific tool adapter (for example many command line tools). V&V Manager plugin automates validation and verification of requirements, system architecture, and behavioural models using OSLC and verification servers, where verification and validation backend tools are installed.

In order to integrate a new verification and validation tool on the verification servers the following process shall be followed:

1. Install the tool on the verification servers.
2. Register the tool on the Proxygen (Facebook's C++ HTTP Libraries) server application. The server needs to know:
    - **The tool binary name to be executed** – only if it is different from the name stated in the OSLC Automation Plan.
    - **The tool parameters** – only if the parameters have to be handled differently than as command line arguments or as a content of a configuration file parameters.
    - **Artefacts under verification** (contracts, requirements, system architecture, system design) – only if the artefacts have to be handled differently than just to be passed as arguments in the form of filenames.

In summary, if the tool binary name, its parameters and the artefacts under verification could be passed to the command line tool in a standard way, the V&V tool does not have to be registered by the verification server application.

2.4 Ad-hoc Tool Integration

The Reuse Company toolset is used as a basis for the study of ad-hoc tool integration. TRC's RQS suite comprises different ad-hoc connectors to enable retrieving requirements from them and run quality assessment processes.

The general idea to create ad-hoc connectors is to identify a suitable API or strategy to exchange information with the desired source. Once this mean is identified, there should be a process to validate it, this can be done revising all the functions needed to retrieve, and specially to send information entity by entity and in sets to speed up the integration.

There have been new additions to this set of ad-hoc connectors in the scope of the AMASS project:

- Integrity: proprietary RMS tool by PTC.
- ReqIF: it is a standard to represent requirements in XML that can be stored in a textual file.
- Rhapsody: even if Rhapsody is focused on modelling, there are requirements being part of those models, so the integration will focus on retrieving them, not the model itself.

2.4.1 ReqIF Connector Integration

This is a new ad-hoc connector created to retrieve and author requirements from ReqIF specifications (see Figure 4).

ReqIF is a well-known standard to represent requirements in XML format. Its structure allows to have several specifications within one project. Indeed, every single ReqIF XML file is considered as a project. Within the project, it may contain 0..N blocks or Specifications. Finally, each Specification may contain both, hierarchically-related Specifications and Objects (requirements). In addition to that, ReqIF allows traceability by creating Relations between Objects within the same ReqIF file.

Despite of the fact that ReqIF is a well-known standard, all the information that is contained within the file is meta-defined. It means that it does not contain fixed attributes to contain the different attributes of the Objects, but contains meta-definition of attributes that are part of the Objects. So that every single attribute is defined in advance within the HEADER of the ReqIF file, and then mapped in the Objects definition.

For that reason, the ReqIF ad-hoc connector needs to pre-define a mapping of the attributes of every single ReqIF Specification to fulfil the RQA/RAT metamodel. This is compulsory to let the tools know where to extract the Statement, Heading, Author, etc., from each ReqIF Object (requirement).

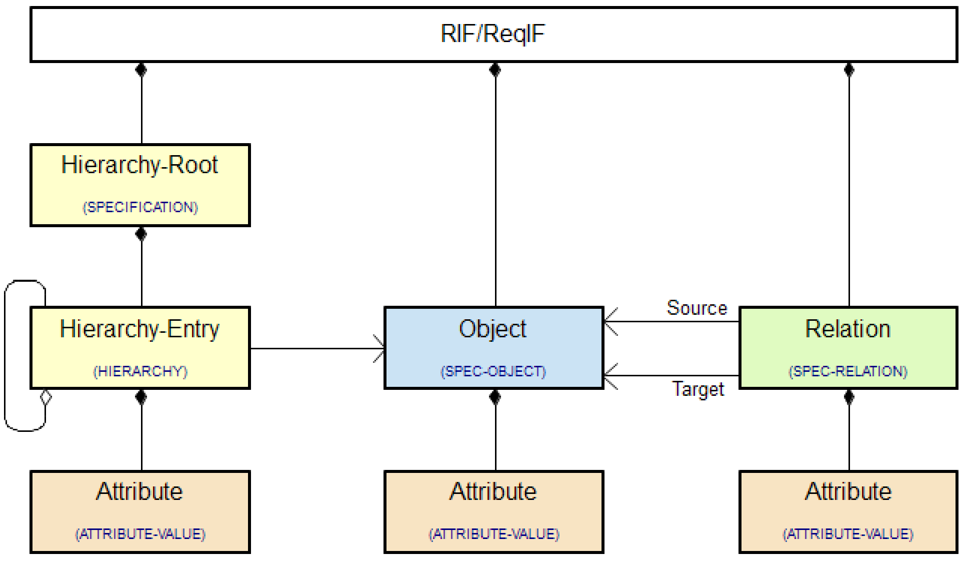


Figure 4. ReqIF metamodel

The strategy used for this ad-hoc integration has been using the programming framework, in this case .NET frameworks for File and XML interoperability.

2.4.2 PTC Integrity Integration

This is a new ad-hoc connector created to retrieve and author requirements from PTC documents.

PTC Integrity follows a typical client/server architecture with the only specific characteristic that the server is a web server composed of many different interfaces. Furthermore, the client has also some possible interactions via its API and coding it in C language.

The integration has been accomplished (Figure 5) by consuming some of these web service interfaces for RQA and RAT tools; and for authoring capabilities on top of the Integrity client (RAT Integrity Plugin), some interactivity has been achieved by using the Integrity client API.

Finally, the integration for the RAT plugin has not been as seamless as done with other RMS tools. Every other RAT plugin has a feature (whose name is RAT Inline), which allows the user to see directly in the requirements grid the quality assessment without opening any other user interface.

The problem arose when understanding the Integrity architecture that the changes are committed to the server and the triggers reacting to these changes were to be executed on the server, that would create an incredible amount of network traffic from RAT Integrity Plugins to the Integrity server and, in addition, the server would be overloaded executing all the trigger actions for all the changes of all the users. However, in other tools, the triggers have the possibility to be handled by the client which is the source of the change that allows to distribute the computing load and to reduce the network traffic to the minimum.

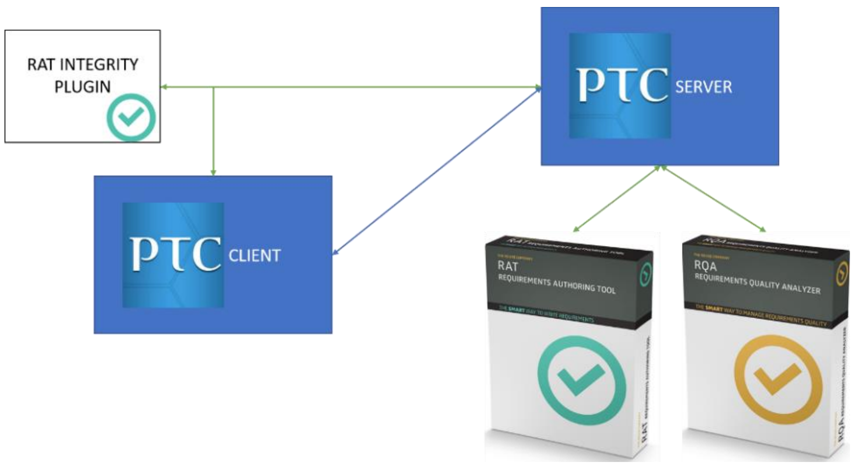

RAT INTEGRITY PLUGIN
PTC SERVER
PTC CLIENT
RAT REQUIREMENTS AUTHORING TOOL
RQA REQUIREMENTS QUALITY ANALYZER

Figure 5. Integrity connector architecture

2.4.3 Rhapsody Integration

This is a new ad-hoc connector created to retrieve requirements from Rhapsody projects. Even if a Rhapsody project is composed of many different models, these models can have requirements related to or inside them, the integration will focus on retrieving and authoring them, not the models themselves.

The Rhapsody architecture is composed of an editing environment working with files stored either locally in the computer or in a network resource. They could also be under management control using any of the well-known version control management tools, such as Git, Subversion, etc.

Rhapsody allows to interoperate with the content of the project using a Java interface as well as other .NET interface, but the later one is obsolete, so it does not allow us to implement our desired functionalities.

The Java interface allows to subscribe handlers to triggers that are fired inside Rhapsody. Then by creating the suitable Java function and subscribing to the desired trigger, any functionality can be implemented.

The integration between RQS tools and Rhapsody has been done using this Java interface. The architecture (see Figure 6) is composed of three different elements:

- RAT Rhapsody plugin: written in Java, subscribes to the suitable triggers in Rhapsody, such as creating and editing requirements, and transfers control to a service (XAT Resident Process) written in .NET and available via a resident process within the same computer.
- The second component of the architecture is written using .NET and consists of two different parts:
  - XAT Resident Process: which it is in charge of using the already existing technology provided by TRC to author requirements via a COM object after receiving any trigger handler.
  - Rhapsody COM interface: it is an interface in charge of communicating the XAT Resident Process and Rhapsody. XAT Resident Process commits the changes performed in the RAT COM object back in Rhapsody via using this interface.
- The third element is the RAT COM object that allows to perform any quality assessment and enables guided authoring using patterns, and makes this functionality also available for other RMS tools plugins.

All these three elements must be deployed in the same computer.

Finally, some major integration points to be mentioned are:

- The requirement format for Rhapsody is HTML and the RQS tool works authoring requirements in RTF format, so a conversion process is performed before using the RAT COM object.
- The RAT COM interface has been improved to allow editing requirements having hyperlinks to any other Rhapsody model element at any position of the requirement.
- RAT Edition window is not possible to be modal on top of Rhapsody with this architecture.

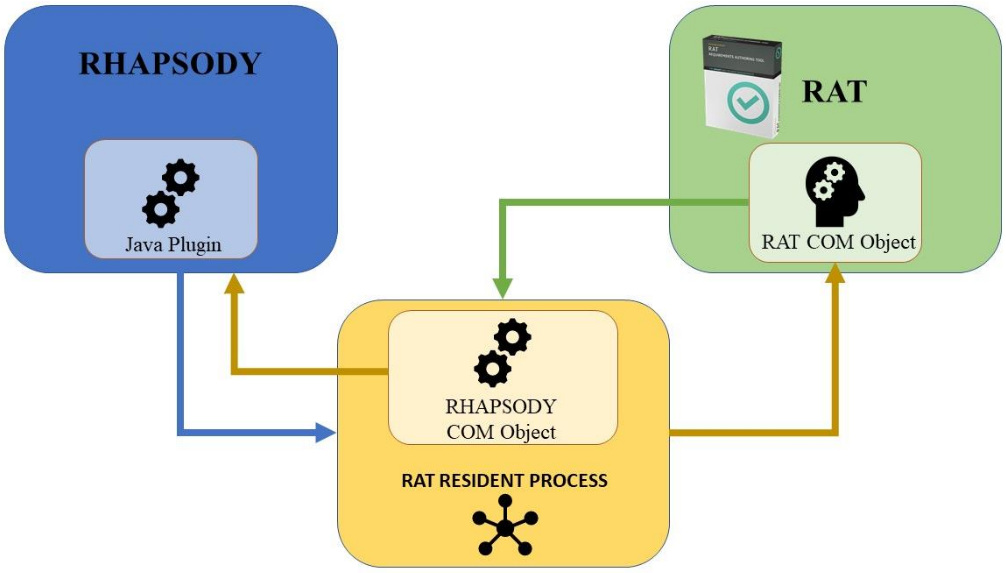


Figure 6. Rhapsody connector architecture

2.5 Papyrus Interoperability

In this section, we describe different ad-hoc connectors to enable exporting and importing model elements from Papyrus. The ad-hoc connectors are available as Papyrus additional components. Note that these components have already been outlined in D5.3 [11]. Here, we recall them and outline their main specifications.

CDO Model Repository integration

Papyrus provides integration with CDO model repository technology for sharing and real-time collaborative editing of Papyrus models. This feature enables to connect to distant repositories using online or offline check-out. The model data are not physically store on the local machine. With the CDO integration, it is possible to create new or open existing Papyrus models in a checkout, it works similar to local workspace projects. One can also import existing model from its local workspace into the distant CDO repository. A functionality helps the user check for cross-references dependencies. It is recommended that any model that reference or is referenced by the initially imported model must also be imported into the repository; otherwise, the model could not be correctly opened if Papyrus is not connected.

RSA integration

Papyrus provides mechanisms to import model elements from RSA/RSA-RTE tool. It supports class, Composite Structure, Object, Profile, Activity, and State Machine diagrams import. The plugin is available in Eclipse Neon. However, it must work on old and newer versions. The integration has been developed in Java and using QvTo transformation.

To import a RSA model into Papyrus, the user must perform the following steps:

1. Copy the .emx or .epx file into an empty (not a Papyrus) project.
2. Use the dedicated "Import RSA Model/Profile" menu to import the corresponding Papyrus model (file *.uml, *.notation and *.di) by right clicking on the RSA files.

To use the Papyrus RSA model importer, the user does not need to have RSA installed on its machine. Further information on this feature is available in the Eclipse Help Content.

Rhapsody integration

Papyrus supports importing SysML Internal Block, Parametric and Block Definition Diagram from Rhapsody tool. The migration tool, done using QvTo language, has been developed with Eclipse Neon and IBM Rhapsody 8.0.3, but it must support previous and next versions.

Because Rhapsody and Papyrus are representing differently similar concepts, the Rhapsody to Papyrus import process is implemented as a set of mapping rules between those two representations. To express the mapping rules, a description of Rhapsody representation of UML and graphical concepts has been implemented in the form of an Ecore metamodel. The metamodel has been built thanks to an analysis of two complementary public information: 1) a public java API providing a first list of the concepts and their inheritance relationship; 2) a list of 150+ examples provided in the Sample directory. Those examples provided a good overview of all the concepts involved in Rhapsody models and how they are serialized in textual files. However, an automated update process is provided as a “developer feature”: when a user provides a new Rhapsody model containing concepts that have not been encountered in the analysed examples, the metamodel update with those new concepts can be automated.

To import the Rhapsody model, the plugin provides the API to convert a ''*.rpy'' into a Papyrus model (''*.uml'', ''*.notation'', ''*.di'' and ''*.properties'' files) following 2 steps:

1. The *.rpy file is converted into a *.umlrpy which is the same model described using the EMF Rhapsody metamodel.
2. the QVTO transformation are automatically called to import the model described in the *.umlrpy file into a Papyrus model (file *.uml, *.notation and *.di).

To use the Papyrus Rhapsody model importer, the user does not need to have Rhapsody installed on its machine. Further information on this feature is available in the Eclipse Help Content.

ReqIF integration

Papyrus provides a traceability solution for connecting SysML elements with ReqIF requirements based on the ReqIF OMG Document Number: formal/2013-10-01 Standard. This feature is implemented in the Papyrus Req tool extension.

To use the import feature, the user must have a SysML model and select the package where ReqIF elements will be imported. The menu "File → Import → Papyrus → Import ReqIF" from Papyrus categories allows to choose the ReqiF file, then the type of requirements to process. A simple user can import instance of requirements inside the Papyrus tool (with relations). An advanced user can, in addition, import new types of requirements inside the Papyrus tool by defining its own profile.

To export the UML elements into a ReqIF, the user must select the model or package where the elements to export are stored. Then, the «File → Export → export ReqIF form the papyrus Categories" menu allows to generate a ReqFile.

Papyrus also offers native features to export and import requirements from excel and csv files, by copy/paste, drag and drop features. See Eclipse Help Content for further details.

Simulink integration

Papyrus has developed a Matlab/Simulink integration (import and export) for co-modelling and co-simulation. The integration aims in one hand at specifying and validating SW functionality. On another hand, it aims at validating the control system performances and define the embedded SW. The plugin is supported by an EMF-based implementation and (QvTo, Acceleo) model-based transformations. It uses Ecore metamodels to generate Stateflow and Simulink Data Dictionary concepts from SysML models and UML state machines. The integration can import/export the models as FMU model for an FMI-based simulation. Hence, other models, e.g., from Dymola, OpenModelica, are also supported.

This feature is not available as Papyrus additional components, but it is provided to the AMASS consortium for free for the duration of the AMASS project.

2.6 V&V Tool Integration

The integration of the V&V tools based on OSLC Automation using V&V Manager is described in section 2.3. Main differentiator is that V&V Manager allows distribution of the verification and validation tasks to multiple servers.

Concerning the V&V Integration with the FBK Tools, two modalities are available: the first one allows to invoke the FBK tools locally by passing the artefacts and the command via files; the second one performs the same functionalities via the OSLC-Automation adapter. It is notable that from the consumer side, these kinds of the integration are almost transparent.

Going more in detail:

Integration of FBK Tool via files

The architecture of the integration towards FBK tools via file is depicted in Figure 7. The tool adapter takes in charge the request from CHESS, converts the model to the Verification tool format, setups the artefacts and the commands files, sends them to the Verification Tools, and in the end returns the result to CHESS, ready to be shown grafically.

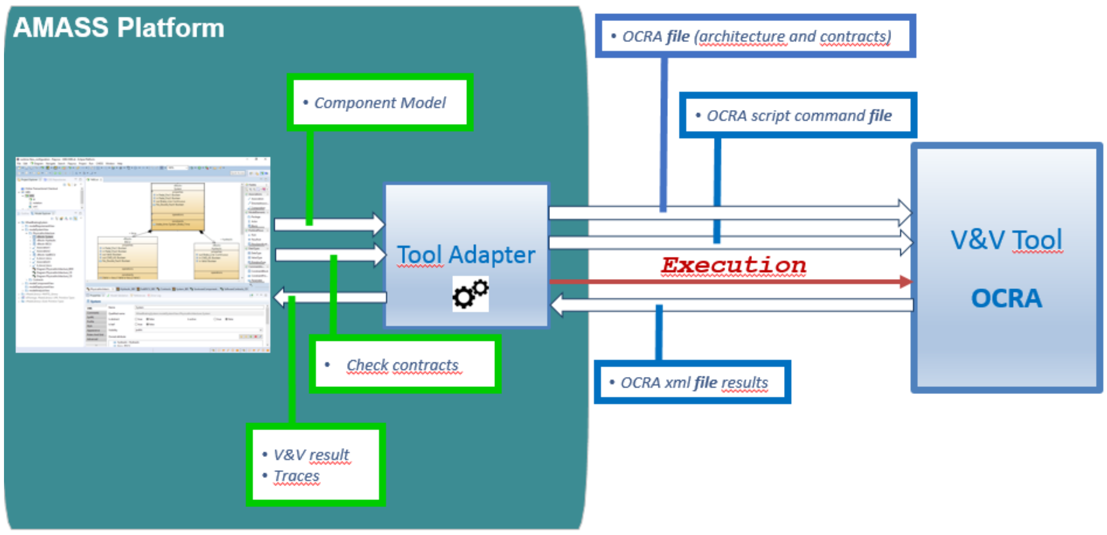


Figure 7. FBK Tool Integration via files

Integration of FBK Tool via OSLC

Figure 8 represents the same functionality described above but using the OSLC approach. As mentioned above, here we choose to use the OSLC Automation Domain for the integration toward the Verification Tools. Each V&V functionality is mapped to an Automation Plan instance, then the Automation Request is set with the parameters (artefact, contract name, properties to be verified, etc.) and finally the Automation Result contains the output of the V&V functionality that has been executed.

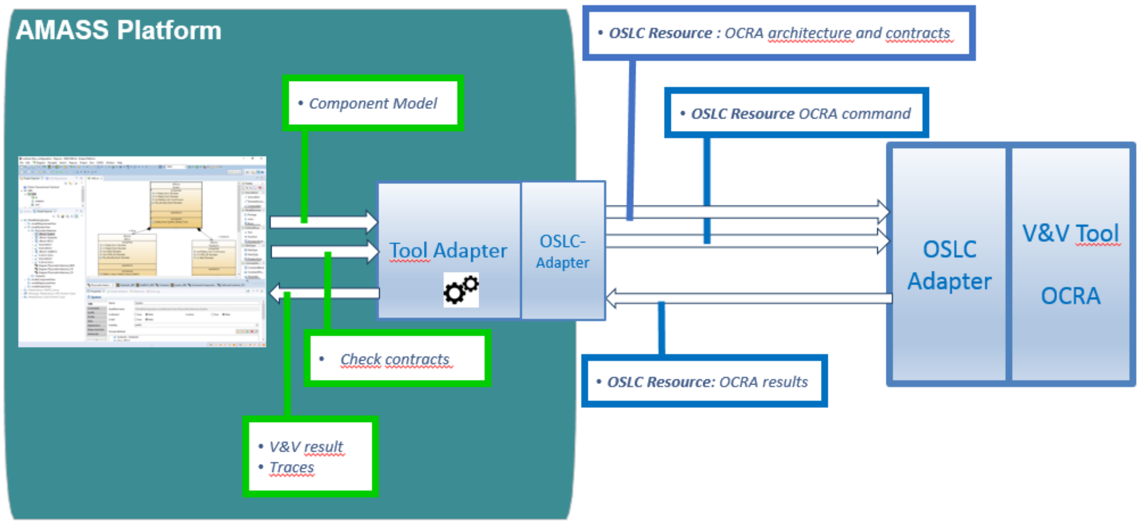


Figure 8. FBK Tool Integration via OSLC Automation

2.7 Seamless Tracing

As discussed in D5.3 [11], in nowadays industrial settings, the safety engineering life-cycle artefacts are the result of various and not integrated tools. As a consequence, seamless traceability, i.e. the relationships between artefacts during the safety engineering lifecycle, represents a serious challenge.

Within AMASS, such challenge is tackled by proposing solutions aimed at overcoming the gaps between different types of safety engineering tools and general-purpose tools. In this deliverable, we briefly recalled the solutions that are expected to be used to guide the AMASS platform users on how to enable seamless tracing.

- The Eclipse project Capra [1] follows the approach of point-to-point integration with only partial data import. Capra aims to provide a modular framework for tracing. Everything besides a small generic core is interchangeable and any number of new trace target types can be defined in new Eclipse plugins.

  There is a conceptual overlap between tracing to external sources and evidence management as the items being traced in a safety case will often (but not always) be used as evidence. It is recommended to trace the appropriate evidence objects, if those are available, for several reasons. Firstly, tracing with evidence gives the traced artefact its semantics, which is important for safety assessments. Secondly, evidence can be updated to a new version by repeating the process that created the artefact. This can be helpful for impact analysis. Finally, it removes redundancies from the model as only one type of artefact or evidence wrapper needs to be created.
- OSLC (as well as OSLC extensions) can be used for the purpose of automatically generating safety cases, and for enabling continuous self-assessment [3], [4], [5], [6] and [7]. This solution could be selected for those sub tool-chains, were OSLC adaptors are available.

2.8 Collaborative Editing

Besides seamless traceability across tool boundaries, the editing and authoring of all kinds of safety case data in teams is a big challenge. Especially when it comes to concurrent (multiple users work independently on the same project data) or even collaborative editing (multiple users work concurrently on the same data). Both aspects become even more challenging when today's diverse tool landscape and IT infrastructures are taken into consideration.

In principle most existing configuration management solutions like Subversion, GIT but also CDO or Google Docs support teamwork in one or another way. Although they are quite different in technology and also in features, they are common - and thus comparable - in the fact that "users create change-sets which are applied to a central data model after a certain delay or time". Note that we treat a change-set here as a side effect of collaboration and not side effect of version management. An example: in Subversion the user creates an offline working copy of the shared data, is editing them (potentially offline) and committing the change set to the Subversion. Time between checkout and commit might be rather long and no connection to the server is required during that time. Google docs is more or less permanently sending small change-sets (so called mutations) to the Google server and changes are applied more or less immediately and by sophisticated merge and transformation logic to the shared data model to avoid conflicts and tedious merges. Figure 9 shows different technologies and their positioning on a XY graph, showing the relation between assumed time between creation and application of a change set and the chance to produce a conflict.

Within AMASS this topic was tackled with prototype solutions to support real time collaboration in core components of the AMASS platform or at least in connected tools. Note that most editing facilities in the AMASS prototypes and also persistency was mainly based on file system (collaboration only with Git/Subversion) or CDO. The main aim was to "move closer to the left", i.e. to offer a solution that "feels more like Google than CDO" but still is applicable to core technologies as EMF/Eclipse.

Conceptually the collaborative real-time editing is based on a simple architecture (see Figure 10):

- There is a server component that maintains EMF models / resources.
- While editing, they are treated as master.
- Clients connect to the server and may retrieve the initial state or may upload an initial state of the EMF model.
- After that, small EMF changes are sent by all clients, applied by the server in the order of appearance (incoming at server) and propagated to all connected clients so they can apply changes from other clients to keep in sync.

- There is no transaction between client and server but changes are applied in sequence and “exclusively” on the server side so any incoming change that produces a conflict (e.g. Due to latency and late income) is rejected and not applied.
- Clients have to communicate to the server to keep in sync.
- Clients may have to revert changes in case they were rejected by the server.

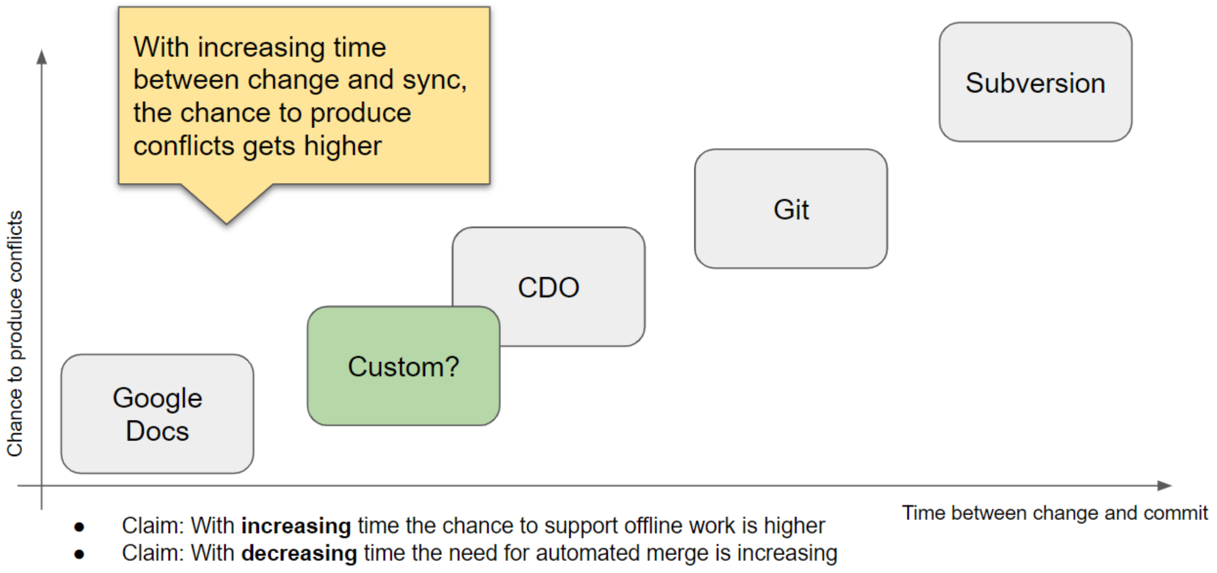


Figure 9. Comparison of revision control systems regarding change sets

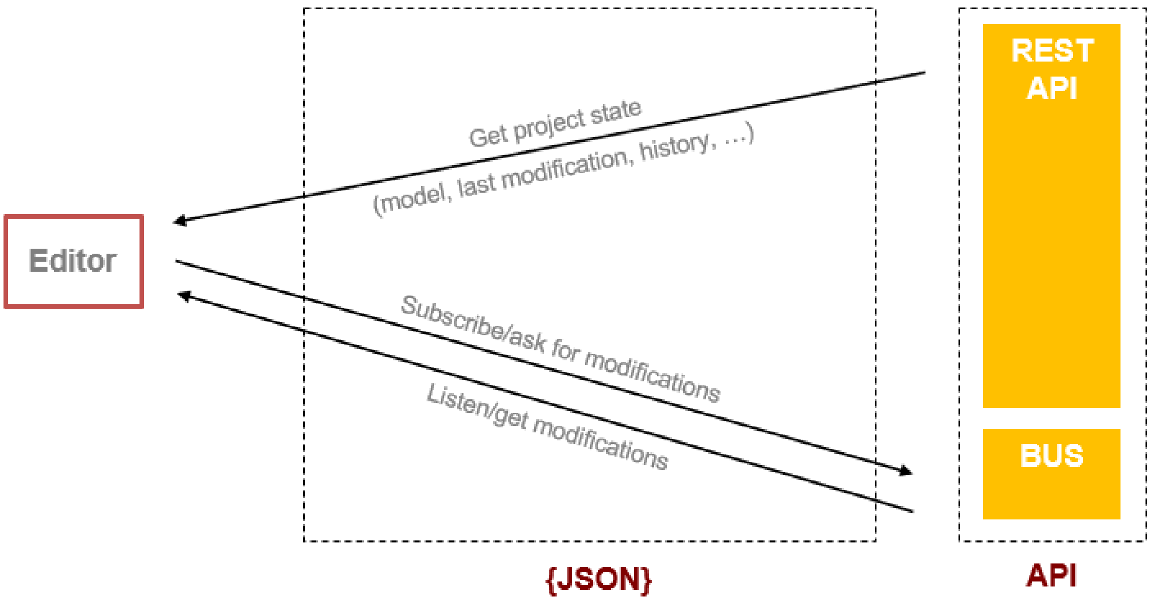

REST API
BUS
Editor
Get project state
(model, last modification, history, ...)
Subscribe/ask for modifications
Listen/get modifications
{JSON}
API

Figure 10. Architecture or centralized collaboration server

This approach is somewhat similar to what was presented earlier in the AMASS project and what is similar to the “EMF Collab” project, which unfortunately seems to be dead. The current prototype is therefore based on a commercial product which offers similar features on a mature state. A lightweight bus is used between server and any client. Clients may register to the bus at any time to send or receive changes. The bus has several “channels”, one is used to exchange change sets, but others are used to manage authentication, data access, browsing etc. A third channel is used to exchange information about which user is doing what at the moment to support collaborative features.

2.9 Safety/Cyber Architect Tools Integration

The integration of the Safety Architect and Cyber Architect tools based on the mapping between CHESS and Safety Architect is described in the deliverable D5.6 [14]. In this deliverable, the solutions that are expected to be used to guide the AMASS platform users is briefly recalled in Figure 11.

The mapping allows the import of a CHESS model scope (Requirement View, System View, Component View, Deployment View and Analysis View) in Safety Architect. The dependability view is also considered during the import. For example, CHESS port failure mode stereotypes are mapped to Safety Architect specific failure Modes. The methodological guidance about how to use Safety/Cyber Architect tools integration is presented in Section 3.8.

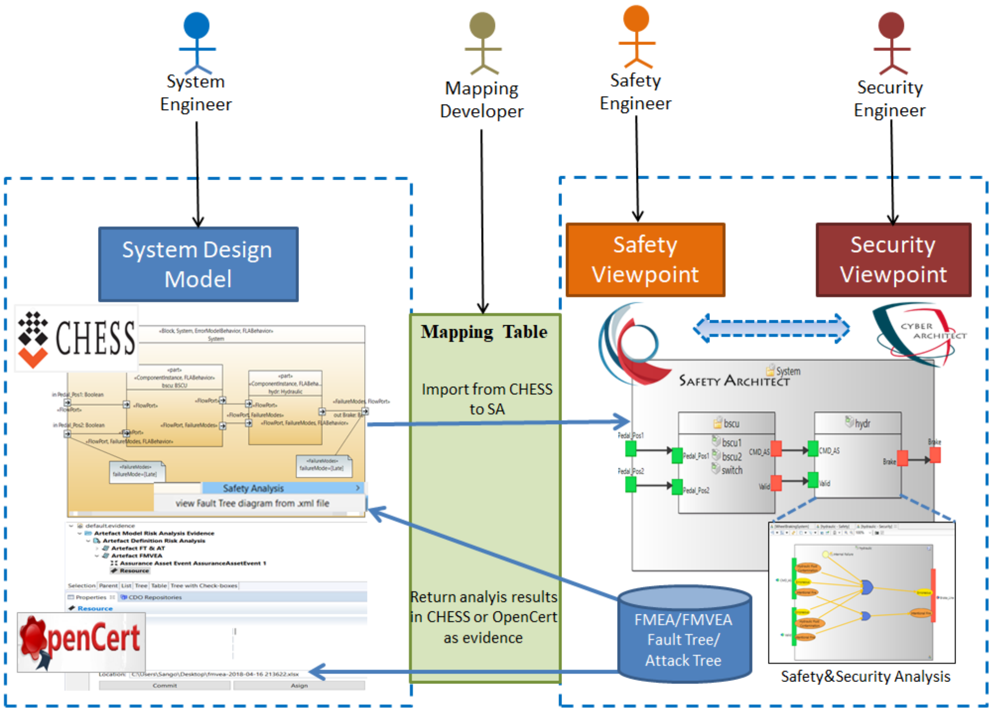


Figure 11. Interoperability between the AMASS platform (CHESS, OpenCert) and Safety/Cyber Architect tools

2.10 Data and Security Management

The Security Management allow creating access policies to models stored in the AMASS CDO Centralised Repository. This access policies will consist in managing users, groups of users and roles:

- **Users** represent a user of the AMASS Platform.
- Users may be grouped together into **Groups of Users**. One user can belong to several groups.
- **Roles** restrict the access and the access rights over the existing data stored by the Data Module in the CDO Repository. A role can be assigned to a user or to a groups or users, that is, to all the members of the group.

The Data Management is responsible of requesting and checking the users access credentials to use AMASS Platform and if they are authorized users, only show to them the data according to their role/s. The Data Management also controls the permissions over the accessible data, avoiding write operation performed by not authorized users and it also restricts the access to certain AMASS Platform functionalities according to the user’s predefined role.

3. Methodological Guide

This chapter presents methodological guidance about how to apply seamless interoperability concepts in AMASS. Each section of this chapter presents a specific technology-dependent guidance that is intended to help in implementing the described technology.

Of course, not all the listed means of interoperability need to be applied in a given situation. In order to also support the selection of the most appropriate technological basis for the developed communication between tools, the *Appendix A. Methodological Guide for Seamless Interoperability – EPF Process Description* is included. The evaluation process to decide which technological section is applicable for the integration of a given tool, based on the requirements on the integration, is described in this appendix in the form provided by the EPF Composer.

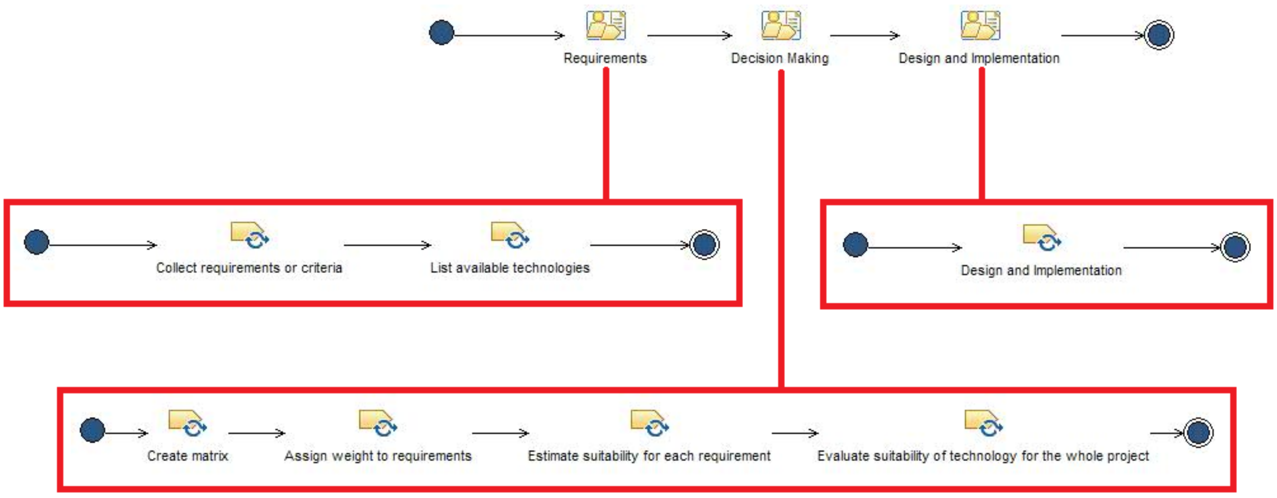

Requirements
Decision Making
Design and Implementation
Collect requirements or criteria
List available technologies
Design and Implementation
Create matrix
Assign weight to requirements
Estimate suitability for each requirement
Evaluate suitability of technology for the whole project

Figure 12. Overview of the development of needed interoperability

Figure 12 is an excerpt of the EPF process description. It contains the suggested workflow for creation of new communication channels between tools. The phases *Requirements* and *Decision Making* should guide the developer to an appropriate section of this Methodological Guide. These two phases can be common to all new interoperability efforts and are described in detail in the Appendix A. Methodological Guide for Seamless Interoperability – EPF Process Description. The third phase *Design and Implementation* has its specific features for each technology. Therefore, the relevant guidance is not included in the general appendix, but it is provided by the individual sections of the chapter.

3.1 Evidence Management

This section presents guidance about how to use evidence-related concepts in AMASS for evidence management in Prototype P1. The section has been divided according to the four main functional areas for evidence management in the AMASS Tool Platform: Evidence Specification, Evidence Traceability, Evidence Evaluation and Evidence Change Impact Analysis.

3.1.1 Evidence Specification

This section provides information about how the artefacts of an assurance project should be specified, focusing on three main aspects.

Artefact Definition

For each instance of a given artefact type (e.g., requirement), an artefact definition must be specified (e.g. Req1, Req2, and Req3). Otherwise, it would not be possible to track their lifecycles independently (e.g.

versions of an artefact definition). The notion of artefact type mainly corresponds to Reference Artefact, but this can vary depending on the nature or purpose of evidence management. Several artefact definitions and their corresponding artefacts can represent the materialisation of a given Reference Artefact.

Artefact granularity

The granularity of the artefacts of an assurance project can vary: set of documents (e.g., system specifications), document (e.g., requirements specification), parts of a document (e.g., a given single requirement), etc. The granularity will depend on the purpose of an artefact and on some traceability-related purposes. As a rule of thumb, an artefact (and thus an artefact definition) must be specified if: (1) the artefact must be linked to others; (2) the lifecycle of the artefact must be tracked, or; (3) the artefact is used in some other general AMASS area (e.g. as evidence for argumentation).

Company- or domain-specific practices

The concepts used for evidence management are generic and aim to support practices across different application domains and companies. However, there exist specific practices that are not directly and explicitly represented in the concepts, and a company must be aware of this. As an example, a company might have its own criteria for evaluating the artefacts of its assurance projects. In this case, evaluation is a broad concept that supports company-specific evaluation practices.

3.1.2 Evidence Traceability

Possible relationships between evidence artefacts include:

- Constrained_By: a relationship of this type from an artefact A to an artefact B documents that artefact B defines some constraint on artefact A, e.g. source code can be constrained by coding standards.
- Satisfies: a relationship of this type from an artefact A to an artefact B documents that artefact A realisation implies artefact B realisation too, e.g. a design specification can satisfy a system requirement.
- Formalises: a relationship of this type from an artefact A to an artefact B documents that artefact A is a formal representation of artefact B, e.g. a Z specification can formalise a requirement specification in UML or natural language.
- Refines: a relationship of this type from an artefact A to an artefact B documents that artefact A defines artefact B in more detail, e.g. a low-level requirement can refine a high-level requirement.
- Derived_From: a relationship of this type from an artefact A to an artefact B documents that artefact A is created from artefact B, e.g. source code can be derived from a system model when a source code generator is used.
- Verifies: a relationship of this type from an artefact A to an artefact B documents that artefact A shows that artefact B properties are true, e.g. model checking results can verify a requirement.
- Validates: a relationship of this type from an artefact A to an artefact B documents that artefact A shows that that artefact B properties can be regarded as valid, e.g. a test case can validate a requirement.
- Implements: a relationship of this type from an artefact A to an artefact B documents that artefact A corresponds to the materialisation of artefact B, e.g. source code can implement an architecture specification.

Two relationships are already explicitly supported in the AMASS Tool Platform: Evolution_Of (precedentVersion) and Composed_Of (artefactPart).

3.1.3 Evidence Evaluation

By evidence evaluation we mainly refer to the activity targeted at judging the adequacy of an artefact and the results associated with this activity. This activity is performed by specifying evaluation events for an artefact and associating the event with a specific evaluation (i.e., its information).

Criteria for evidence evaluation can include:

- Completeness: unknown, incomplete, draft, final, obsolete.
- Consistency: unknown, informal, semiformal, formal; other options could be unknown, consistent, inconsistent, or unknown, informally consistent, semi-formally consistent, formally consistent.
- Originality: unknown, derivative, original.
- Relevance: unknown, low, mediumLow, medium, mediumHigh, high.
- Reliability: unknown, unReliable, nonUsuallyReliable, usuallyReliable, fairlyReliable, completelyReliable
- Significance: unknown, low, mediumLow, medium, mediumHigh, high.
- Strength: a numerical value between 0 and 100.
- Trustworthiness: unknown, low, mediumLow, medium, mediumHigh, high.
- Appropriateness: unknown, low, mediumLow, medium, mediumHigh, high.

As mentioned above, a company can have its own evidence evaluation criteria. The most common approach in industry for evidence evaluation is the use of checklists, thus conformance to a checklist or to some of its items can be used as evaluation criterion.

3.1.4 Evidence Change Impact Analysis

Evidence change impact analysis can be necessary in the different general situations listed below. This impact analysis can be triggered by the changes in different artefact types mentioned, which can also be affected by changes. The insights below are based on the results of a large industrial survey [2].

The situations in which the respondents had to deal with evidence change impact analysis are:

- Modification of a new system during its development
- Modification of a new system as a result of its V&V
- Reuse of existing components in a new system
- Re-certification of an existing system after some modification
- Modification of a system during its maintenance
- New safety-related request from an assessor or a certification authority
- Re-certification of an existing system for a different operational context
- Re-certification of an existing system for a different standard
- Re-certification of an existing system for a different application domain
- Changes in system criticality level
- Independent assessment of the risk management process
- Hazards identified after the fact
- Re-certification for temporary works
- Accident analysis
- System of system reuse

Regarding the artefact types involved in evidence change impact analysis, Table 2 shows the median frequency (5-point Likert scale: never, few projects, some projects, most projects, and every project) with which different artefact types trigger impact analysis and are affected by changes in the body of the safety evidence.

Table 2. Role of different artefact types in evidence change impact analysis

	Impact Analysis Trigger	**Affected by Changes**
System Lifecycle Plans	Some projects	Few projects
Reused Components Information	Few projects-Some projects	Few projects
Personnel Competence Specifications	Few projects	Few projects
Safety Analysis Results	Most projects	Some projects
Assumptions and Operation Conditions Specifications	Some projects	Some projects
Requirements Specifications	Most projects	Some projects
Architecture Specifications	Some projects	Some projects
Design Specifications	Most projects	Some projects
Traceability Specifications	Most projects	Some projects
Test Case Specifications	Most projects	Some projects
Tool-Supported V&V Results	Some projects	Few projects
Manual V&V Results	Some projects	Most projects
Source Code	Most projects	Some projects
Safety Cases	Some projects	Some projects

3.2 OSLC KM

Following, the methodology to apply the OSLC KM approach will be presented through a case study based on a Linked Data architecture in which different software components and tools are integrated to provide a service for software reuse. This case is presented following the next steps:

1. Motivation and rationale
2. Selection of software artefacts and tool providers
3. Implementation of a Linked Data architecture for integration and interoperability
4. Results and discussion
5. Research Limitations and Lessons Learnt

3.2.1 Motivation and Rationale

An organization developing a cyber-physical system, a rugged computer, is looking for a solution to integrate all tools involved in the development lifecycle. Instead of using a complete ALM or PLM suite, they follow a decentralized and federated approach where different tool providers can be found.

They use a requirements management tool (RMT) for gathering and storing stakeholder and system requirements. These requirements are written using boilerplates in combination with a requirements quality checking tool to ensure correctness, consistency and completeness. They also have tools for software (UML) and dynamic systems modelling. Besides, changes and issues that can occur during the project are also registered and managed as well as the test cases and their results. A continuous integration server is another tool that provides to the development team a way to monitor changes in source control and trigger any failure during the build-phase. Finally, the software artefacts that are released are expected to be reused in other products.

In this context, the organization is looking for the best way to integrate and reuse all the artefacts that are being continuously generated. Currently, ad-hoc integrations are being made. For instance, the requirements quality checking connects to the RMT through an interface that offers requirements under a non-standard protocol and data model. Moreover, and due to the cost of the licenses, the organization is seeking new RMT providers but, so far, they have not made any decision due to the integration costs of the new tool in the development lifecycle.

On the other hand, it is not possible to unify names in all artefacts that are generated. Sometimes requirements contain entities that do not appear in the models or test cases, preventing the possibility of recovering traceability links. Thus, the cost of reusing any existing artefact is becoming higher due to the fact that is not possible to completely trace a component from its inception to the final release. In conclusion, this organization is facing the following issues:

1. Lack of a product breakdown structure to drive the development lifecycle.
2. Name mismatches.
3. Point-to-point and ad-hoc integrations between a client and a tool provider.
4. A plethora of heterogeneous protocols and data models (most of them non-standard ones).
5. Impossibility of reusing artefacts since traces cannot be recovered.
6. Lack of a software knowledge repository to store and search for artefacts (metadata and contents).
7. Poor documentation mechanisms. Lack of graphical view of artefact dependencies.
8. Standalone applications not ready for a collaborative web environment.
9. Vendor lock-in.

Due to all these reasons, they are interested in a holistic and standard approach that can tackle these issues, easing the development lifecycle and boosting the reuse of existing and future artefacts.

3.2.2 Selection of Software Artefacts and Tool Providers

Building on the previous scenario, some tools have been identified to carry out the functionality required in the development lifecycle, see a summary in Table 3:

- **Concept and Requirement Pattern.** A domain vocabulary comprising concepts and relationships is used to ensure that requirements only contain domain terminology. A domain expert has devised this vocabulary, as well as the product breakdown structure in which the composition of the product under development is formalized. The creation of a domain vocabulary requires a great effort in terms of time and human resources. That is why a domain vocabulary is expected to be reused in further developments. On the other hand, a requirement pattern is defined as a sequence of restrictions with place-holders for specific terms and values. It constitutes a particular knowledge statement that can be syntactically and semantically described using the domain concepts and relationships. The use of patterns for easing the writing of requirements is a widely used practice that helps engineers to avoid inconsistent specifications. Both, concepts and requirement patterns, are knowledge items that are stored in the Knowledge Manager tool. This tool implements the OSLC KM specification and knowledge items can now be accessed as web information resources following the Linked Data principles.
- **Quality Metrics.** More specifically, requirements quality metrics are a set of quantitative measures that serve engineers to verify the quality of a requirements specification (consistency, completeness and correctness). They are based on the INCOSE (International Council on Systems Engineering) guidelines for writing good requirements. The current set comprises 58 different metrics that are mainly focused on consistency. These metrics have been implemented in the Requirements Quality Analyzer (RQA) tool, and an OSLC interface has been implemented to expose these metrics through a REST and RDF interface.
- **Change/Issue.** Tracking of changes and issues in a software project is a key activity for allowing individuals or teams of developers to keep track of existing bugs in their software products. Most software project repositories, such as Github or Bitbucket, contain a service for tracking issues. In this case study, Bugzilla has been selected as change/issue service provider. This service has been already wrapped in an OSLC interface implementing the OSLC Change Management 2.0 specification.

- **Test.** This is a kind of artefact generated during the development lifecycle for testing software at different levels and with different purposes. Commonly tests are executed in a test plan to ensure that a program, module or interface with an associated control data is operating in the proper way. There are different types of testing techniques such as unit testing, performance testing, functional testing or black and white box testing. In this case study, the IBM Quality Manager application within the IBM Jazz Platform has been selected for accessing test cases and plans. This service implements the OSLC Quality Management 2.0 specification.
- **Software Model.** A conceptual model is a model comprising different concepts, relationships and documentation. Models are used to represent people's knowledge or understanding about a particular domain or situation. UML provides a graphical notation to draw diagrams to represent software models. Basically, it is possible to have static (e.g. classes) and dynamic (e.g. states) diagrams. In this case study, the OSCL adapter on top of the Magic Draw tool (a UML 2.0 metamodel compliant tool) has been selected to access UML models. This OSLC layer has been released as part of the work within the OSLC MBSE Working Group at OMG.
- **Dynamic System Model.** It is a particular type of model that represents relationships through mathematical equations. For instance, an electric circuit model is a kind of Dynamic System Model used to define and simulate the behaviour of a circuit. The Modelica language, a non-proprietary, object-oriented, equation-based language to model complex physical systems, has been selected to define dynamic system models. In this case, there is neither OSLC specification nor formal ontology to represent and share dynamic system models. That is why the OSLC KM specification will be used for these purposes, boosting the interoperability among tools in the development lifecycle.
- **Automation Resource.** In the context of OSLC, an automation resource is a kind of resource managed by a service. They are usually part of an automation plan that generates some result. As an example, the use of a tool for continuous integration can be seen as a service that automatically manages a set of resources (e.g. source code, third-party libraries or test cases) for building a project (result). In this case study, the Jenkins/Hudson CI service and the OSLC adapter implementing the OSLC Automation Resource 2.0 specification have been selected for demonstration purposes.

According to this configuration, there is a set of OSLC-based services that are expected to consume and provide different domains or types of artefact. In the specific case of dynamic system models, the Modelica language has not been promoted to the OSLC initiative yet, so any client aiming at consuming a Dynamic System Model should implement its own parser and interpreter of the language. The main advantages of having a common data shape for any knowledge item or software artefact seems clear when no shape is already defined or when a common software knowledge repository is required (i.e. when information must be stored under the same model).

Table 3. Artefacts, tool provider and OSLC adapters

ID	Type of Artefact	Tool (Provider)	OSLC Domain	OSLC Adapter
1	Concept and Requirement Pattern	knowledgeMANAGER (The Reuse Company Inc.)	Knowledge Management (OSLC KM)	OSLC4NET[5]
2	Requirement	IBM DNG (Doors Next Generation) (IBM)	Requirement Management (OSLC RM)	Eclipse Lyo[6]
3	Quality metric	Requirements Quality Analyzer (The Reuse Company Inc.)	Estimation and Measurement metrics (OSLC EMS)	OSLC4NET
4	Change/Issue	Bugzilla (Mozilla)	Change Management (OSLC CM)	Eclipse Lyo[7]
5	Test	IBM DNG Quality Manager (IBM)	Quality Management (OSCL QM)	Eclipse Lyo[8]
6	Software Model	Magic Draw (No Magic Inc.)	MBSE (OSLC KM)	Eclipse Lyo[9]
7	Dynamic System Model	Open Modelica (Open Modelica Association)	Not available	Not available
8	Automation Resource	Jenkins/Hudson CI (Jenkins CI)	Automation (OSLC AM)	Eclipse Lyo[10]

3.2.3 Implementation of a Linked Data architecture for Integration and Interoperability in a Software Reuse Environment

In order to design and implement a Linked Data architecture it is necessary to take into account the federated and distributed character of information and services. In this light, there is an increasing interest in the creation of methodologies, best practices/recipes and lifecycles, design of URIs, design patterns, publication of RDF datasets and vocabularies and establishment of Linked Data profiles. In particular, the proposed Linked Data architectures correspond to a REST architecture where service providers exchange data about information resources; artefacts in the case of software reuse. Data is generated *on-the-fly* and the design of URIs and data management is delegated in these third-party services.

To do so, the aforementioned service providers for the different OSLC domains are depicted in Figure 13. Every provider offers an interface that is compliant with the W3C Linked Data Platform Recommendation (OSLC APIs have been tested for this purpose in the recommendation). In this specification, a Linked Data Platform Container is a collection of information resources, in this case, artefacts. They are offered as Linked Data Platform and RDF resources, so it is possible to access and manage them through HTTP protocols (REST-based fashion). Thus, it is possible to easily exchanged artefacts data between the different consumers and providers. From a logical point of view, this architecture is based on a set of federated services that are deployed in a distributed environment.

[5] https://oslc4net.codeplex.com/
[6] https://jazz.net/library/article/1382
[7] http://wiki.eclipse.org/Lyo/BuildBugzilla
[8] https://jazz.net/products/rational-quality-manager/
[9] http://wiki.eclipse.org/Lyo/MagicDraw
[10] https://wiki.eclipse.org/Lyo/JenkinsPlugin

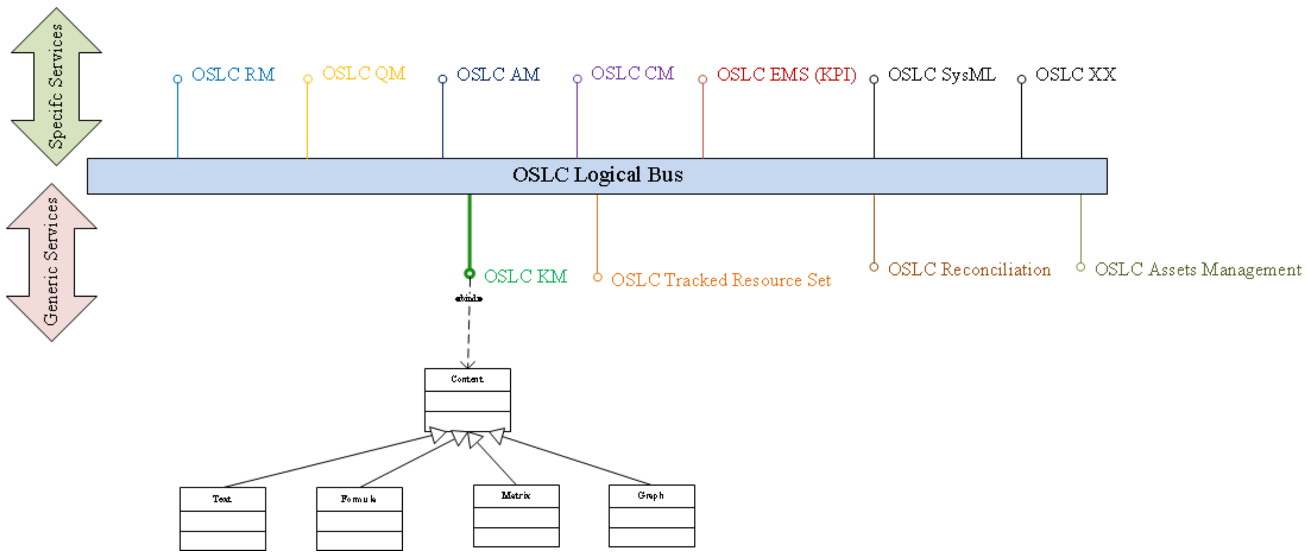
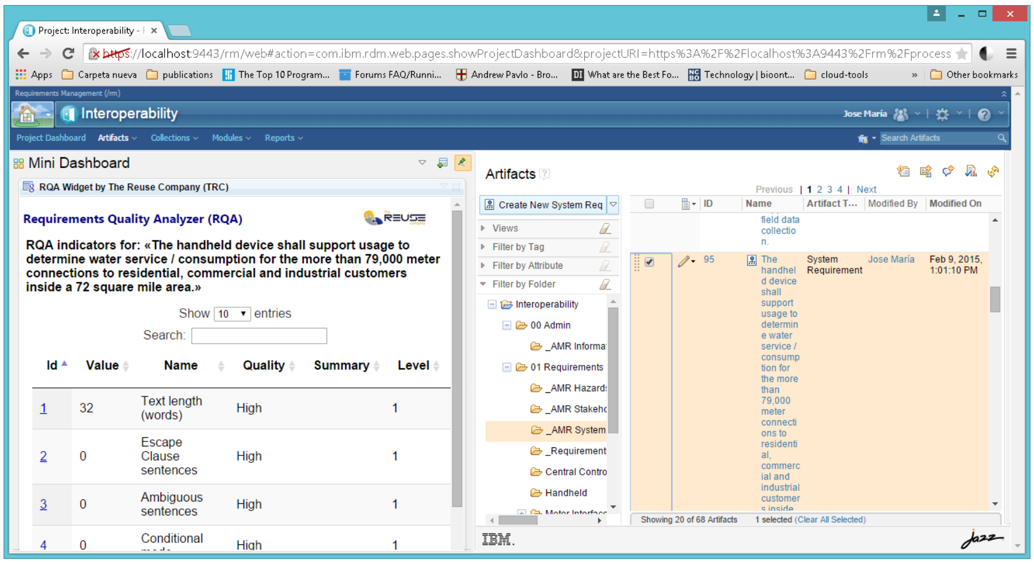


Figure 13. A federated and distributed architecture of OSLC-based services and providers

In this particular case study, the OSLC KM provider consumes the information resources from the specific services layer and offers an interface to index and search artefacts. Following, a scenario-based approach is used to briefly explain some of the possibilities of this architecture.

- **Requirements quality checking.** Figure 14 depicts the visualization of requirements quality metrics within the Jazz Platform. The client takes a selected requirement in IBM DNG (OSLC RM), sends a request to the OSLC EMS (Key Performance Indicators-KPI) service and receives the set of quality metrics. To calculate these metrics, the OSLC EMS provider (RQA) also requests to the OSLC KM the set of concepts, requirements patterns and the product breakdown structure. Thus, it is possible to establish the quality of a requirement according to a set of vocabulary-based metrics. On the other hand, the OSLC KM provider consumes the information about a requirement to offer an index and retrieval service. The requirement is also linked to a test (OSLC QM) or an issue (OSLC CM) and this information can be retrieved through the search interface of the OSLC KM.



RQA Widget by The Reuse Company (TRC)

Requirements Quality Analyzer (RQA)

RQA indicators for: «The handheld device shall support usage to determine water service / consumption for the more than 79,000 meter connections to residential, commercial and industrial customers inside a 72 square mile area.»

Show 10 entries

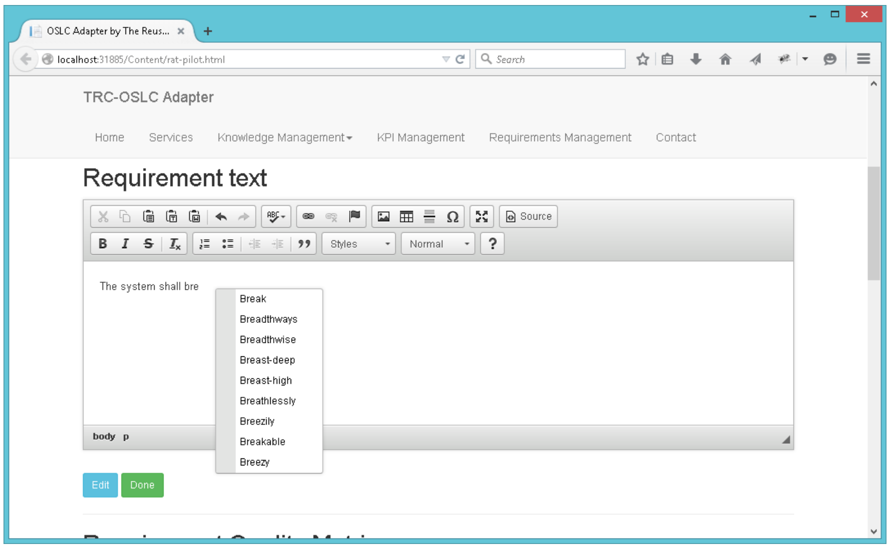
Search:

Id	Value	Name	Quality	Summary	Level
1	32	Text length (words)	High		1
2	0	Escape Clause sentences	High		1
3	0	Ambiguous sentences	High		1
4	0	Conditional	High		1

Figure 14. Integration between IBM Doors DNG and Requirements Quality Metrics through OSLC

- **Requirements authoring.** As it has been previously presented, the use of patterns is a common practice to guide the writing of requirements. Figure 15 shows the creation of an "IntelliSense" service within the CK Editor (a web-based editor). The client requests the set of patterns through OSLC to the OSLC

KM provider. Then, patterns are used to constrain the textual description of the requirement offering a new context-aware text completion feature.



The system shall bre

Figure 15. Implementation of "IntelliSense" capabilities through OSLC in CK Editor

- **Indexing and retrieval of non-OSLC artefacts**. In this third case, there is an electrical circuit created with the Open Modelica Editor, see Figure 16. Since there is not OSLC specification for representing such information, the OSLC KM data shape can be used instead. A processor of the Modelica language has been implemented to transform the electrical circuit into an OSLC KM data shape that is now available in the Knowledge Manager, see Figure 17. Thus, it is possible to index, link and search the circuit in the software knowledge repository reusing the capabilities of the Knowledge Manager and exposing results through the OSLC KM interface. Furthermore, this approach can be followed for any piece of (RDF-encoded) data.

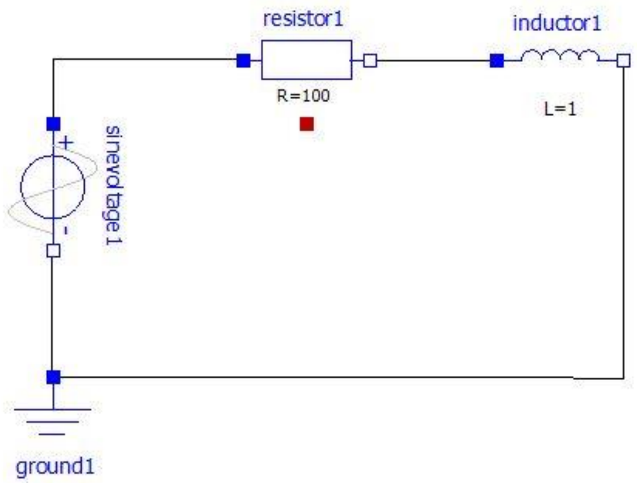
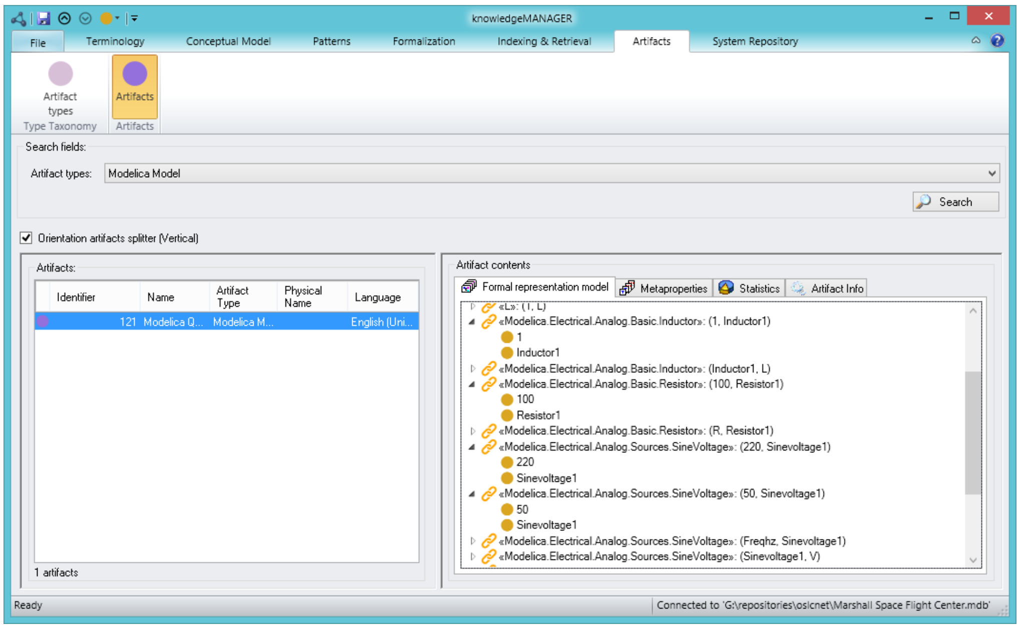


Figure 16. A Low-pass filter circuit edited in Open Modelica



Search fields:

Artifact types: Modelica Model

☑ Orientation artifacts splitter (Vertical)

Artifacts:

	Identifier	Name	Artifact Type	Physical Name	Language
●	121	Modelica Q...	Modelica M...		English (Uni...

1 artifacts

Artifact contents

```
▷ «L»: (1, L)
◢ «Modelica.Electrical.Analog.Basic.Inductor»: (1, Inductor1)
    1
    Inductor1
▷ «Modelica.Electrical.Analog.Basic.Inductor»: (Inductor1, L)
◢ «Modelica.Electrical.Analog.Basic.Resistor»: (100, Resistor1)
    100
    Resistor1
▷ «Modelica.Electrical.Analog.Basic.Resistor»: (R, Resistor1)
◢ «Modelica.Electrical.Analog.Sources.SineVoltage»: (220, Sinevoltage1)
    220
    Sinevoltage1
◢ «Modelica.Electrical.Analog.Sources.SineVoltage»: (50, Sinevoltage1)
    50
    Sinevoltage1
▷ «Modelica.Electrical.Analog.Sources.SineVoltage»: (Freqhz, Sinevoltage1)
▷ «Modelica.Electrical.Analog.Sources.SineVoltage»: (Sinevoltage1, V)
```

Ready | Connected to 'G:\repositories\oslcnet\Marshall Space Flight Center.mdb'

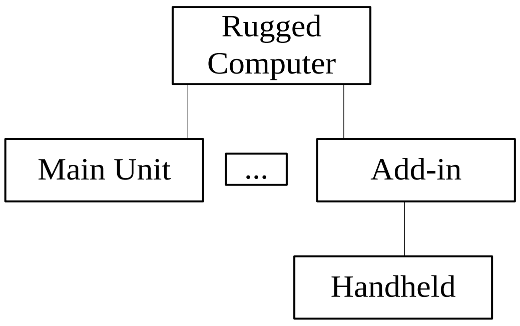
Figure 17. The Low-pass filter circuit Artefact indexed in Knowledge Manager

Building on the abovementioned scenarios, Listing 8 presents some simplified RDF code snippets[11] of the different OSLC domains as Linked Data (coloured links). Sub-Figure 18 presents a simple product breakdown structure that can be encoded in OSLC KM as Sub-Listing 3 shows. There is also a system requirement for the handheld component that is presented in Sub-Listing 4. This requirement contains links to the elements in the PBS, a link to a change request that has been registered in Sub-Listing 6 and a link to the physical circuit that physically implements part of the handheld functionality, see Sub-Listing 7. On the other hand, it is possible to check the quality of requirements. A quality metric regarding the number of words is depicted in Sub-Listing 5 and linked to the handheld component in the PBS.

Finally, and since the different artefacts generated during the development lifecycle are linked together building an underlying knowledge graph, it is possible to perform select queries through the OSLC query capabilities (depending on the service provider), an SPARQL interface or even natural language (if Knowledge Manager is used). For instance, if we want to reuse all components without defects and which requirements quality metrics are high (having more than 30 words in the indicator "number of words"), we can formulate the next SPARQL query, see Listing 9 (in this example SPARQL is used as kind of abstract syntax).

In conclusion, this explanation through a case study has presented a real use case in which different artefacts are generated during the development lifecycle. Following the OSLC and Linked Data principles, information resources (artefacts) are exposed through a REST interface and can be gathered and integrated within a common repository. This repository also offers an interface (OSLC KM) that eases the search of existing artefacts and representation (metadata and contents) of those artefacts that do not have a RDF representation. Thus, it is possible to enhance reuse capabilities of artefacts since all of them are represented under the same paradigm.

[11] The prefixed of the different RDF vocabularies have been gathered from the prefix.cc service.



Sub-Figure 18. Preferred visualization and PBS of a "Rugged Computer"

```
:device a oslc_km:Artefact;
  oslc_km:preferred-visualization
<URI_TO_FIGURE_11>;
  oslc_km:interpretation :PBS.
  oslc_km:observations :RuggedComputerKPI;
  oslc_km:term :device_term.

:device_term a oslc_km:Term, skos:Concept
       skos:prefLabel "Rugged Computer"@en;
       skos:narrower :handheld;
       skos:closeMatch
<http://dbpedia.org/resource/Rugged_computer>.

:handheld a oslc_km:Artefact;
  oslc_km:term :handeld_term.

:handeld_term a oslc_km:Term, skos:Concept;
 skos:prefLabel "Handheld"@en.

:rsph1 a oslc_km: Relationship;
 oslc_km:from :car;
 rdfs:label "has-part";
 oslc_km:to:braking_system.
```

Sub-Listing 3. Partial example of a PBS and a controlled vocabulary as OSCL KM artefacts.

```
:r1 a oslc-rm:Requirement;
 dcterms:title  "The  handheld  device
shall support usage to determine water
service/ consumption for the more than
79,000    meter    connections    to
residential, commercial and industrial
customers  inside  a  72  square  mile
area";
…

oslc_rm:uses :device;
oslc_rm:uses :handheld;
oslc_rm:affectedBy :c1;
oslc_rm:implementedBy :f1;
.
```

Sub-Listing 4. Partial example of a requirement following the OSCL RM specification.

```
:o1     a       qb:observation ;
 qb:dataset :RuggedComputerKPI;
 sdmx-concept:obsStatus
 sdmx-code:obsStatus-E;
 :ref-artefact :r1.
 :ref-indicator :NWords.
 :value "32"^^xsd:double .
…

:NWords a Ios_kpi:KPI
…
dcterms:title "Number of Words".
```

Sub-Listing 5. Partial example of an observation in the OSLC EMS (KPI) vocabulary

```
:c1 a oslc-cm:ChangeRequest;
 dcterms:title  "Defect  detected  in
handheld to properly determine water
service";
 oslc_cm:status "Closed";

oslc_cm:affectsRequirement :r1;
dcterms:subject :device;
dcterms:subject :handheld;
oslc_cm:tracksChangeSet <GIT_URI>;
…
.
```

Sub-Listing 6. Partial example of a change request following the OSCL CM specification.

```
:f1 a oslc_km:Artefact;
  oslc_km:preferred-visualization
<URI_TO_FIGURE_9>;
  oslc_km:term :f1_term.

:f1_term a oslc_km:Term, skos:Concept
       skos:prefLabel "Lower Pass Filter"@en;
…
.
```

Sub-Listing 7. Partial example of the lower pass filter as and OSLC KM artefact.

Listing 8. Software reuse environment through Linked Data (RDF code snippets in Turtle syntax)

```
SELECT ?component WHERE{
 ?component rdf:type oslc_km:Artefact.
 :device skos:narrower ?component.
 ?requirement rdf:type oslc-rm:Requirement.
 OPTIONAL {
  ?requirement oslc_rm:affectedBy ?change.
  ?change oslc_cm:status ?status.
  FILTER regex(?status,'^Closed,'i').
 }
 ?observation  :ref-artefact ?component.
 ?observation  :ref-indicator :NWords.
 ?observation  :value ?value.
 FILTER (?value >= 30).
}
```

Listing 9. SPARQL query to gather components without defects and with high-quality requirements

3.2.4 Results and Discussion

The development of a Linked Data-based architecture following the guidelines of OSLC has generated several positive effects. The aforementioned issues that the organization under study was facing are now mitigated, see Table 4. This is mainly due to the use of standards and a common data shape for knowledge management. Although some of the artefacts in this case study are beyond software reuse they are part of usual development lifecycles.

3.2.5 Limitations and Lessons Learnt

Some key limitations of the presented work must be outlined. The first one depends on the number and type of services and defined resources. This case study has been conducted in a closed world and, more specifically, eight different types of artefacts and service providers have been tested.

A new OSLC domain, OSLC KM, has been defined and implemented for knowledge management. This domain takes inspiration from existing W3C recommendations so that, in a broader and real scope, this specification could change to meet real industry requirements.

In the same manner, new service providers and domains are expected to be integrated in this case study to ensure the representation capabilities of the OSLC KM specification. However, this work presents an industry-oriented case study based on a real environment for software and knowledge reuse.

Building on the previous comment, we cannot figure out either the internal budget, methodologies, tools, domain vocabularies, experience and background of particular organizations. We merely observe and re-use existing public and on-line knowledge sources to provide a demonstrative case study of an OSLC-based architecture for software reuse.

On the other hand, it seems clear that after a long time, software reuse is an active research area in which a good number of challenges and open issues can be found. The emerging application of the OSLC principles to enable interoperability among tools in the development lifecycle is providing a new opportunity for enhancing software reuse techniques. Assuming that interoperability will be reached in terms of data models and protocols for exchanging artefacts, it is necessary to provide data models for representing both metadata and contents of any artefact.

Table 4. Issues in the case study and mitigating factors

ID	Issue Description	Mitigation
1	Lack of a product breakdown structure (PBS) to drive the development lifecycle.	The possibility of defining and sharing a PBS under the OSLC initiative enables practitioners the management of complex processes in the development lifecycle.
2	Name mismatches.	A common domain vocabulary can now be shared and reused in other tools.
3	Point-to-point and ad-hoc integrations between a client and a tool provider.	This is a common side-effect of reusing standards and software knowledge repository. Consumers can ask the repository for a particular artefact, and once the metadata (e.g. information access) and contents are gathered, they can directly request data to the real provider.
4	A plethora of heterogeneous protocols and data models (most of them non-standard ones).	Although each tool can have its internal data model, there is a unified and shared input/output interface based on RDF.
5	Impossibility of reusing artefacts since traces cannot be recovered.	Having the possibility of representing any artefact under the same data model can help to recover traces.
6	Lack of a software knowledge repository to store and search for artefacts (metadata and contents).	Any piece of software or knowledge can now be represented using concepts and relationships.
7	Poor documentation mechanisms. Lack of graphical view of artefact dependencies.	As a side-effect, the implementation of the OSLC KM specification on top the Knowledge Manager provides also a mechanism for visualizing artefacts or even generating documentation templates.
8	Standalone applications not ready for a collaborative web environment.	The use of services in a federated architecture enables practitioners the deployment of applications in different locations making the development lifecycle more flexible and scalable. On the contrary, performance, security and privacy issues can emerge avoiding the proper development a collaborative environment for software development.
9	Vendor lock-in.	The use of a standard layer for exchanging data and information avoids a complete vendor lock-in. It is possible to easily change the provider of a service if it also implements that particular OSLC specification.

In this light, RDF has demonstrated to be a very good candidate as an input/output data model. Actually, one of the main and well-known drawbacks of RDF is that just a few tools natively work in RDF. However, other languages such as RDFS or OWL, designed for representing logical statements, lack of the proper constructors to represent any piece of knowledge based on other paradigms. Although it is possible to define a RDFS or OWL vocabulary for a particular domain, the reality is that most of the time domain experts do not really need an underlying formal logic but a flexible language for representing concepts and relationships. In this sense, the use of SRL as a language to represent metadata and contents of any artefact has been demonstrated to be flexible and practical (including native tool support).

In the context of data validation, previous section (motivation) has outlined the increasing interest of checking data consistency and integrity of RDF graphs. In software reuse environment, this approach can be applied to matchmaking of software artefacts.

From a technical point of view, the deployment of the tools and OSLC adapters has involved a major technical challenge, due to the need of configuration for every tool vendor and adapter. That is why we consider that new trends in micro services should be applied to decrease the time to deploy and test. Furthermore, an OSLC-based architecture for software reuse also requires a new mindset to move existing applications to a web environment in which context issues regarding authorization or authentication are completely different.

On the other hand, a federated and distributed environment of services also implies potential issues regarding security, privacy and performance. That is why this new paradigm must be carefully managed to avoid well-known problems such as information loss, bottlenecks or denegation of service attacks, to name just a few.

Finally, the OSLC initiative is continuously releasing and updating specifications, some of them have been already promoted to OASIS standards. This also means that the industry support and commitment behind of OSLC is strongly encouraging interoperability through the creation and use of standards.

3.2.6 Use in the AMASS platform

The application of the OSLC KM approach, presented in the previous sections, inside the AMASS platform has created an outcome of adding new functions that allows to seamlessly incorporate new evidence specifications from other different sources that can be generated during the development lifecycle.

The idea is to populate the project in the AMASS repository with all types of information that can be relevant while trying to certificate the Cyber-Physical System as evidences.

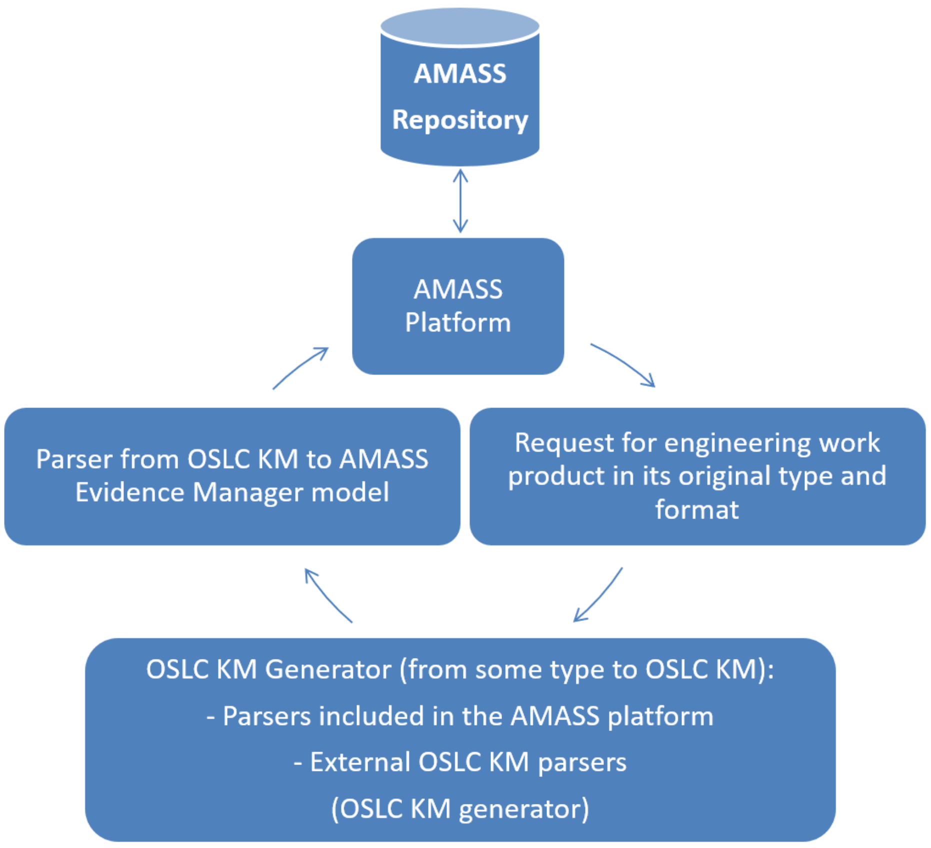

AMASS Repository
AMASS Platform
Parser from OSLC KM to AMASS Evidence Manager model
Request for engineering work product in its original type and format
OSLC KM Generator (from some type to OSLC KM):
- Parsers included in the AMASS platform
- External OSLC KM parsers
(OSLC KM generator)

Figure 19. The Evidence Manager import process using OSLC KM

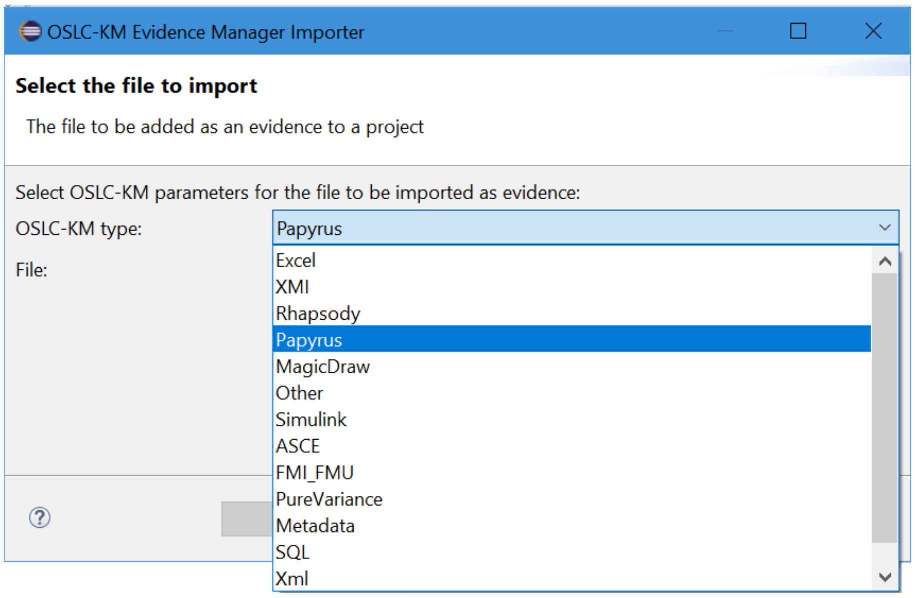
For example, if the system is required by some standard, to have gone through a requirement quality analysis process. The result of the quality analysis generated using RQA tool by TRC can be incorporated in the project in the AMASS repository by transforming its output into a new OSLC KM instance. Now, this

OSCL KM model instance, can be easily added to the project in the AMASS repository by the mapping created between the OSLC KM model and the AMASS Evidence Manager model.

In general, these development lifecycle engineering sources can be found in many different types and formats.

In the frame of AMASS project, as can be seen in Figure 20, a set of OSLC KM parsers has been developed to map such evidences in their native type and format with the AMASS Evidence Manager:

- Microsoft Excel
- Standard XMI (output from many UML tools)
- SysML from Rhapsody
- SysML from Papyrus
- SysML from Magic Draw
- SysML from Other tool providers
- Simulink
- ASCE
- FMI/FMU
- Pure Variance
- Metadata
- SQL
- XML
- SRL encoded in JSON format


OSLC-KM Evidence Manager Importer

Select the file to import

The file to be added as an evidence to a project

Select OSLC-KM parameters for the file to be imported as evidence:

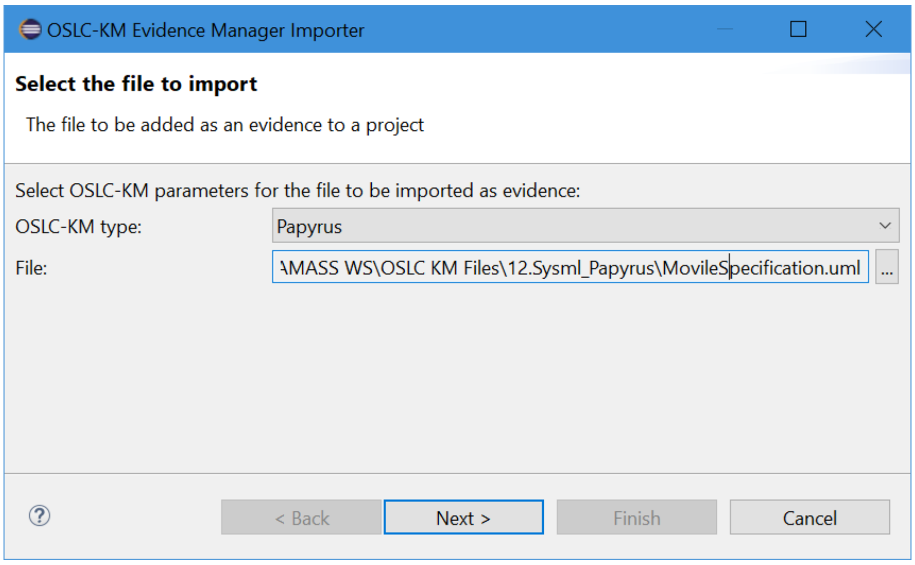
OSLC-KM type: Papyrus

File:

Excel
XMI
Rhapsody
Papyrus
MagicDraw
Other
Simulink
ASCE
FMI_FMU
PureVariance
Metadata
SQL
Xml

Figure 20. The OSLC-KM Evidence Manager Importer Wizard showing the available OSLC-KM parsers

If any other connector should be needed, the only task needed to be executed is the creation of the OSLC KM model from the information retrieved in the original source of the product of the development lifecycle.


OSLC-KM Evidence Manager Importer

Select the file to import

The file to be added as an evidence to a project

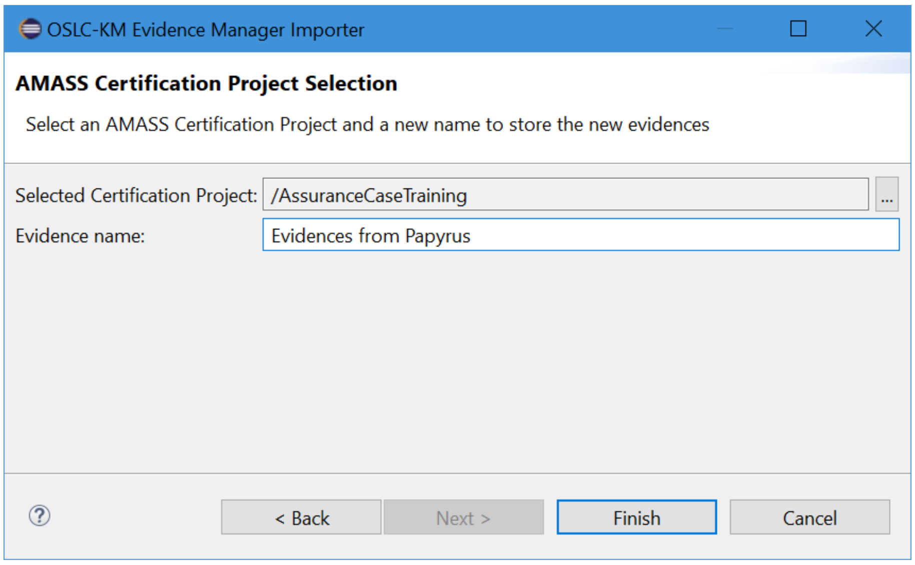
Select OSLC-KM parameters for the file to be imported as evidence:

OSLC-KM type: Papyrus

File: \AMASS WS\OSLC KM Files\12.Sysml_Papyrus\MovileSpecification.uml

Figure 21. The OSLC-KM Evidence Manager Importer Wizard showing a connection to a Papyrus model

And, finally the parser that transforms any instance of the OSLC KM model in the AMASS Evidence Manager model has been incorporated in the AMASS platform.


OSLC-KM Evidence Manager Importer

AMASS Certification Project Selection

Select an AMASS Certification Project and a new name to store the new evidences

Selected Certification Project: /AssuranceCaseTraining

Evidence name: Evidences from Papyrus

Figure 22. The OSLC-KM Evidence Manager Importer Wizard selecting the AMASS project to store the evidences

With this last parser, all the pieces needed to perform the process of importing new evidences to the project in the AMASS repository have been completed.

3.3 V&V Manager and OSLC Automation

Integration based on OSLC Automation offers data and process integration of the tools, while any interactive or control integration is not possible since these cannot be automated. This section describes the solution provided by V&V Manager plug-in that allows seamlessly distribute verification and validation of artefacts (requirements, system design, etc.) to verification servers.

Optionally, the integration of FBK V&V Tools based on OSLC Automation could be used without the V&V Manager as described in Section 3.6.

3.3.1 SysML Elements and Corresponding OSLC Properties

Table 5 shows how the SysML elements from CHESS Requirements are mapped to OSLC properties of OSLC Requirement resource (http://open-services.net/ns/rm#Requirement), which is referenced from OSLC Automation Plan. This OSLC Automation Plan will be sent by V&V Manager to verification servers.

Table 5. Mapping of SysML elements to OSLC properties for OSLC Requirement resource

SysML element	OSLC property	Occurs	Value-type	Description
text	dcterms:description	zero-or-one	XML Literal	Descriptive text (reference: Dublin Core) about Requirement resource. V&V Manager expects text in the form of FormalProperty
id	dcterms:identifier	zero-or-one	String	An identifier for a Requirement resource. This identifier may be unique with a scope that is defined by the RM provider. Assigned by the service provider when a resource is created. Not intended for end-user display.
/derived	oslc_rm:decomposes	zero-or-many	Reference	The object (Requirement) is decomposed by the subject (Requirement).
/derivedFrom	oslc_rm:decomposedBy	zero-or-many	Reference	The subject (Requirement) is decomposed by the object (Requirement).
/satisfiedBy	oslc_rm:satisfiedBy	zero-or-many	Reference	The subject is satisfied by the object. For example, a user requirement is satisfied by a system requirement.
/refinedBy	oslc_rm:specifiedBy	zero-or-many	Reference	The subject is specified by the object. For example, a requirement is refined by model or more refined requirement.
/tracedTo	oslc_rm:trackedBy	zero-or-many	Reference	Resource, such as a change request, which tracks this requirement.
/verifiedBy	oslc_rm:validatedBy	zero-or-many	Reference	Resource, such as a test case, which validates this requirement.
Author	dcterm:creator dcterm:contributor	zero-or-many	AnyResource	Creator or creators/contributor or contributors of Requirement resource (reference: Dublin Core). It is likely that the target resource will be an foaf:Person but that is not necessarily the case.

SysML element	OSLC property	Occurs	Value-type	Description
<currently missing>	dcterms:created	zero-or-one	DateTime	Timestamp of Requirement resource creation (reference: Dublin Core)
<currently missing>	dcterms:modified	zero-or-one	DateTime	Timestamp of latest Requirement resource modification (reference: Dublin Core)
Type (Functional, Mission, Interface, Environmental, Physical, Operational, Human Factor, Logistics support, Configuration, Design, Verification, Product Assurance)	rdf:type	zero-or-many	Resource	The Requirement resource type URIs.
• Status (Initial, Derived, Final) • Priority (High, Medium, Low) • Maturity (TBC, TBD, Is analysis, Analyzed) • Risk (High, Medium, Low) • /master	oslc_auto:parameterDefinition	zero-or-many	AnyResource	The definition of a parameter for this Automation Plan. parameterDefinitions are either a local (inline) or referenced resource and use the attributes (the range) of the oslc:Property resource

Similarly, Table 6 shows OSLC properties of Automation Plan OSLC resource and information of who created the automation plan by calling the V&V Manager and when.

Table 6. Mapping of SysML elements to OSLC properties for OSLC Automation Plan resource

SysML element	OSLC property	Occurs	Value-type	Description
Author	dcterm:creator dcterm:contributor	zero-or-many	AnyResource	Creator or creators/contributor or contributors of resource (reference: Dublin Core). It is likely that the target resource will be an foaf:Person but that is not necessarily the case.
Created	dcterms:created	zero-or-one	DateTime	Timestamp of resource creation (reference: Dublin Core)

3.3.2 Public Verification Server

While proprietary production verification servers allow distribution of verification and validation tasks, Masaryk University created a publicly available verification server that hosts multiple verification and validation tools that are accessible using OSLC Automation: http://pleiada01.fi.muni.cz:8080

Automation server as implemented by Honeywell is based on Facebook's C++ HTTP Libraries – Proxygen.

There are several verification and validation tools for requirement semantic analysis installed:

- Tools for requirement semantic analysis, in particular logical consistency, redundancy, and vacuity checking:
    - **Looney** and **Remus2** for consistency and redundancy checking – created by Masaryk University (including computation of Minimal Unsatisfiable Subsets)
- Tools mainly used for realizability checking:
    - **Acacia+** - a tool for solving the LTL realizability and synthesis problems
    - **autoCode4** – an engine that synthesizes controllers from formal specifications described under a subset of LTL.
    - **Party Elli** – SMT based Bounded Synthesis
    - **BoSy** – a reactive synthesis tool based on constraint-solving
- Model checking tool:
    - DIVINE – explicit state model checker
- Supporting tools:
    - **Z3** Theorem Prover
    - **SPOT** – a C++14 library for LTL, ω-automata manipulation and model checking.

Tools for formal verification of system architecture and system design will be installed till the release of P2 prototype.

3.4 Ad-hoc Tool Integration

When dealing with the task of creating a new integration with a tool (Figure 23) the goal is to instantiate a standard model within the application, in this case the meta-model of AMASS from the desired tool. If the integration cannot be achieved by means of any standard, an ad-hoc strategy is the only way to do it.

The general idea to create an ad-hoc integration is to identify a suitable API or strategy to exchange information with the desired source. Once this mean is identified, there should be a process to validate it, this can be done revising all the functions needed to retrieve, and specially to send information entity by entity and in sets to speed up the integration.

If this mean of communication fulfils all the functionality needs and its performance meets the expectations, for example in comparison with other integrations and the user experience in the application, then a new ad-hoc connector can be created.

Our recommendation to create the ad-hoc connector is to use an iterative approach. From our expertise the minimum iterations to be executed are:

1. Create a façade library to retrieve the information in the programming technology desired using the selected strategy. E.g. in our case for Integrity we have built a library in .NET exposing functions to RQA application and when executing them consuming the Integrity WS available in the Integrity Server.
2. Create a configuration connection set to gather all the information either from the user or predefined settings.

3. Instantiate the façade with all the configuration connection parameters, connecting physically to the source and retrieving the essential minimum pieces of information by executing the essential functions. E.g. in our case retrieving the structure of project, modules and requirements.
4. Fill up all other parts of the model with the rest of functions available in the façade.

From our expertise, the ad-hoc integration is an evolutive process, the initial releases always fulfil the initial need of connecting to the source, but after some feedback from real users, a better understanding of the architecture created on top of the connected source, indicates that some changes or improvements must be done to our ad-hoc connector. Then the iterative process described is repeated to create a new release.

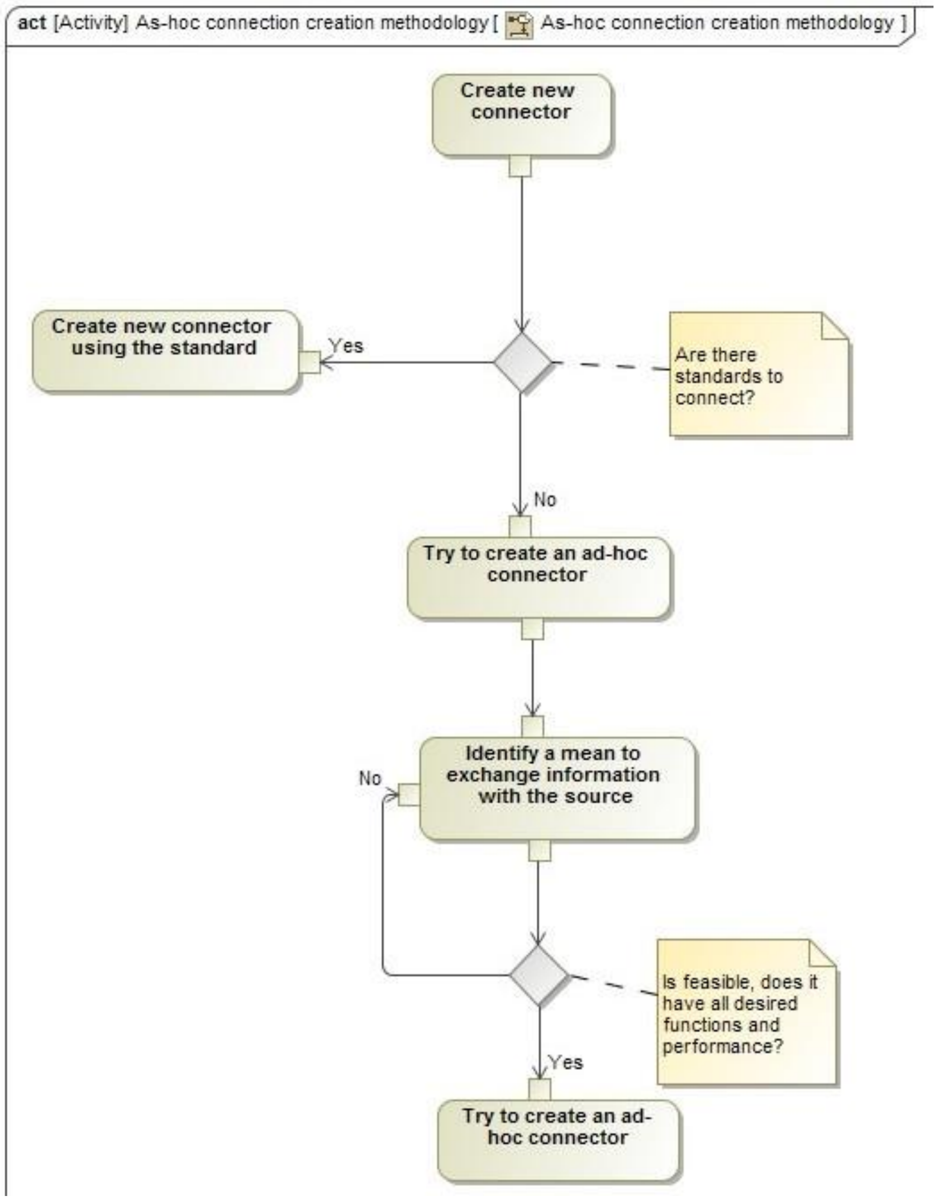
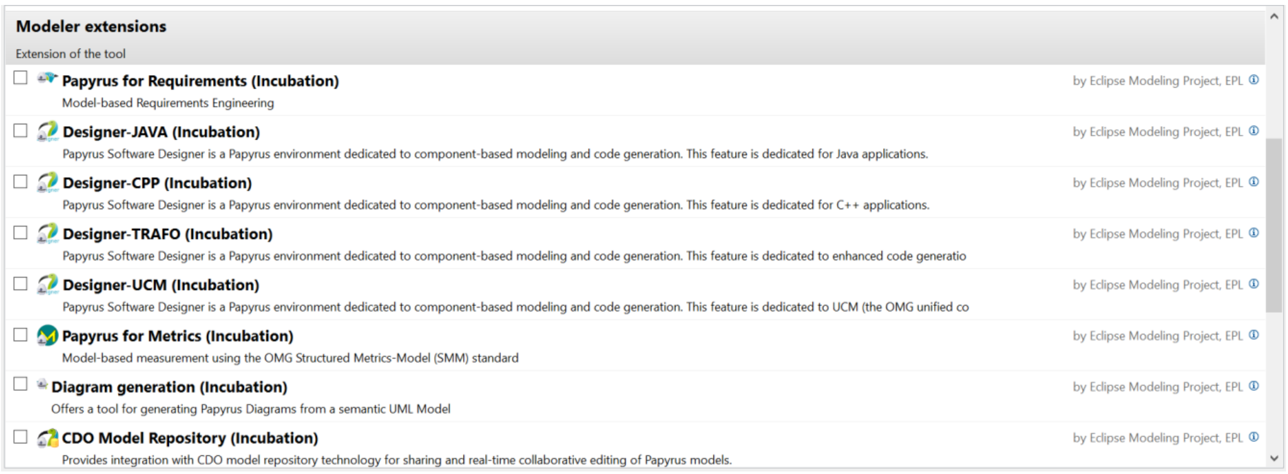

act [Activity] As-hoc connection creation methodology [As-hoc connection creation methodology]
Create new connector
Create new connector using the standard
Yes
Are there standards to connect?
No
Try to create an ad-hoc connector
Identify a mean to exchange information with the source
No
Is feasible, does it have all desired functions and performance?
Yes
Try to create an ad-hoc connector

Figure 23. Ad-hoc connection methodology process

3.5 Papyrus Interoperability

The Papyrus extra components are not provided by default in Papyrus. The CDO model repository, RSA, Rhapsody and ReqIF integration features are provided as Papyrus extra components. To install these features, one must open the Papyrus discovery wizard from the **Help → Install Papyrus Additional Components** menu and select the desired entry from the presented list (Figure 24). One can select many entries at a time, and then press **Finish** to perform the installation. It is recommended to restart Eclipse to integrate the changes in the environment. Others Papyrus are available as independent plugins, e.g. the model-based simulation tool, that can be download from Papyrus development repositories.



Modeler extensions
Extension of the tool

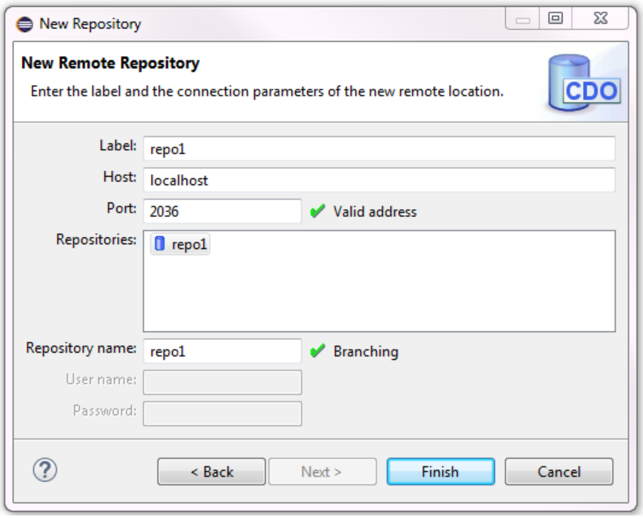
- ☐ **Papyrus for Requirements (Incubation)** — by Eclipse Modeling Project, EPL
  Model-based Requirements Engineering
- ☐ **Designer-JAVA (Incubation)** — by Eclipse Modeling Project, EPL
  Papyrus Software Designer is a Papyrus environment dedicated to component-based modeling and code generation. This feature is dedicated for Java applications.
- ☐ **Designer-CPP (Incubation)** — by Eclipse Modeling Project, EPL
  Papyrus Software Designer is a Papyrus environment dedicated to component-based modeling and code generation. This feature is dedicated for C++ applications.
- ☐ **Designer-TRAFO (Incubation)** — by Eclipse Modeling Project, EPL
  Papyrus Software Designer is a Papyrus environment dedicated to component-based modeling and code generation. This feature is dedicated to enhanced code generatio
- ☐ **Designer-UCM (Incubation)** — by Eclipse Modeling Project, EPL
  Papyrus Software Designer is a Papyrus environment dedicated to component-based modeling and code generation. This feature is dedicated to UCM (the OMG unified co
- ☐ **Papyrus for Metrics (Incubation)** — by Eclipse Modeling Project, EPL
  Model-based measurement using the OMG Structured Metrics-Model (SMM) standard
- ☐ **Diagram generation (Incubation)** — by Eclipse Modeling Project, EPL
  Offers a tool for generating Papyrus Diagrams from a semantic UML Model
- ☐ **CDO Model Repository (Incubation)** — by Eclipse Modeling Project, EPL
  Provides integration with CDO model repository technology for sharing and real-time collaborative editing of Papyrus models.

Figure 24. Excerpt of Papyrus Additional components Discovery view

CDO model repository integration

To connect to a CDO repository (Figure 25), the user must follow the steps:

- Use the green plus (+) button in the in the CDO Repositories view.
- Fill in the information to create a new repository. A new repository is added in the view.
- Select the new repository and pick Checkout in the context menu.
- Complete the wizard. A new checkout representing the contents of the repository now appears in the Project Explorer view.



New Repository

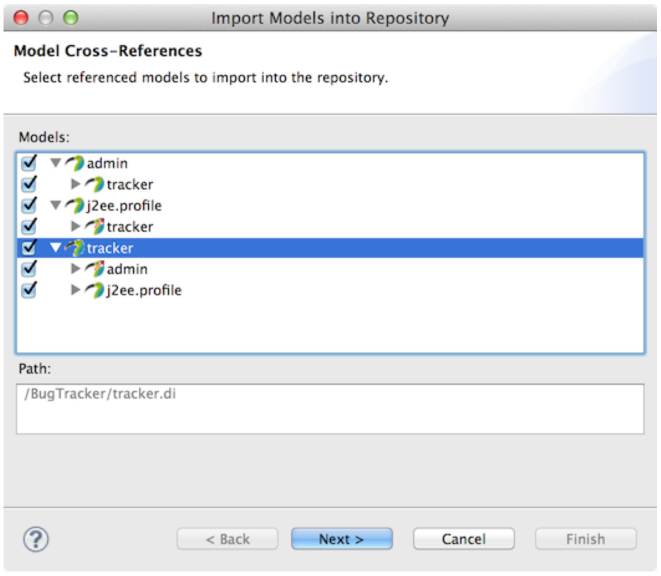
New Remote Repository
Enter the label and the connection parameters of the new remote location.

Label:	repo1
Host:	localhost
Port:	2036 ✓ Valid address
Repositories:	repo1
Repository name:	repo1 ✓ Branching
User name:	
Password:	

Figure 25. How to connect to a CDO repository

To import models in CDO repository (Figure 26), the user must follow the steps:

- Select one or more models in the Project Explorer.
- Choose the Import into Repository... action in the context menu. Alternatively, just drag and drop one or more models onto a repository.
- By default, the wizard maps incoming models to paths in the repository according to their paths in the workspace. This mapping may be customized in the last page.


Import Models into Repository

Model Cross-References

Select referenced models to import into the repository.

Models:

- ✓ admin
  - ✓ tracker
- ✓ j2ee.profile
  - ✓ tracker
- ✓ tracker
  - ✓ admin
  - ✓ j2ee.profile

Path: /BugTracker/tracker.di

Figure 26. Import model into CDO repository

RSA integration

Here are the steps to import the RSA models into Papyrus:

- Create a new, empty general project.
- Copy the RSA .epx (profile) and .emx (model) files into the empty project folder.
- Import the RSA model (.emx file) into Papyrus by right-clicking on the .emx file and then select "Import RSA Model/Profile". As a result, the RSA .emx file is replaced by the Papyrus model file.
- Do the same for the .epx file. As a result, the Papyrus profile model file is created.
- Replace RSA Profile by Papyrus Profile by changing the pointer from the .epx (RSA) file to the .uml (Payprus) file. To do so:
  - Close Papyrus.
  - Replace all occurrences of "ITU-T_protocol-neutral-model_profile.epx" by "ITU-T_protocol-neutral-model_profile.profile.uml" in the model .uml file using any ASCII editor.
  - Reopen Papyrus. The model is ready to be used.

Rhapsody integration

Here are the steps to import the Rhapsody models into Papyrus:

- Create a new, empty general project.
- Copy the Rhapsody *.rpy file into the empty project folder.
- Right click on the .rpy file and then select "Import Rhapsody model". The *.rpy file is converted into a *.umlrpy.
- The QVTO transformation are automatically called to import the model described in the *.umlrpy file into a Papyrus model (file *.uml, *.notation and *.di).

ReqIF integration

To use the ReqIF integration feature in Papyrus, the kind of User, either user or advanced has to be specified first. To do so, use the menu "window--> preferences--> Papyrus--> ReqIF Import".

To import a ReqIf file (Figure 27), the user must follow the steps:

- Create a SysML model and select a package within the model.

- Select the menu Import of Eclipse.
- Select the menu Import ReqIF from Papyrus Categories.
- Select the ReqIF file to import.
- Select Requirement types of ReqIF file that you want to import.
- Or create the profile that will contain imported types for advanced user.
- The model has now imported requirements with relations.

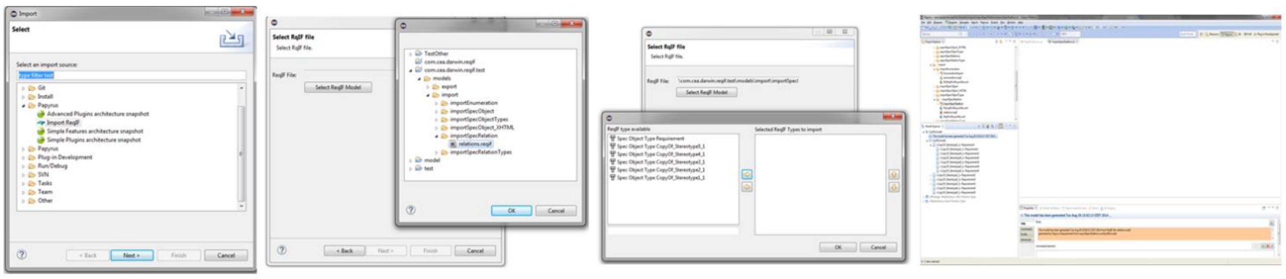
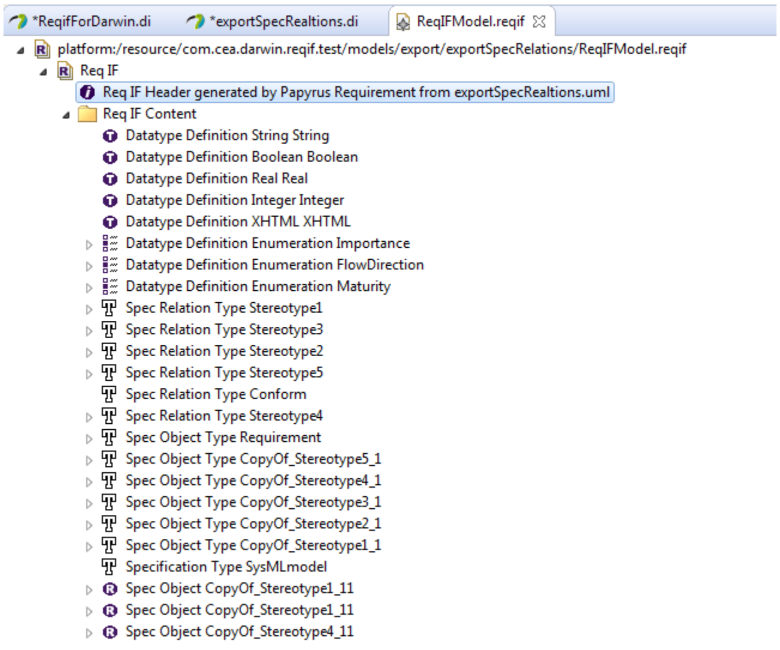


Figure 27. Steps to Import ReqiF file into Papyrus model

To export the Sysml requirements into a ReqIf file (Figure 28), the user must perform the following steps:

- Select the SysML model that you want to export.
- Select the export Menu from Eclipse menu.
- Select the menu export ReqIF form the papyrus Categories.
- Choose the name of the reqif file. A reqFile is generated.



```
platform:/resource/com.cea.darwin.reqif.test/models/export/exportSpecRelations/ReqIFModel.reqif
  Req IF
    Req IF Header generated by Papyrus Requirement from exportSpecRealtions.uml
    Req IF Content
      Datatype Definition String String
      Datatype Definition Boolean Boolean
      Datatype Definition Real Real
      Datatype Definition Integer Integer
      Datatype Definition XHTML XHTML
      Datatype Definition Enumeration Importance
      Datatype Definition Enumeration FlowDirection
      Datatype Definition Enumeration Maturity
      Spec Relation Type Stereotype1
      Spec Relation Type Stereotype3
      Spec Relation Type Stereotype2
      Spec Relation Type Stereotype5
      Spec Relation Type Conform
      Spec Relation Type Stereotype4
      Spec Object Type Requirement
      Spec Object Type CopyOf_Stereotype5_1
      Spec Object Type CopyOf_Stereotype4_1
      Spec Object Type CopyOf_Stereotype3_1
      Spec Object Type CopyOf_Stereotype2_1
      Spec Object Type CopyOf_Stereotype1_1
      Specification Type SysMLmodel
      Spec Object CopyOf_Stereotype1_11
      Spec Object CopyOf_Stereotype1_11
      Spec Object CopyOf_Stereotype4_11
```

Figure 28. Exported Papyrus model into ReqIF file

3.6 V&V Tool Integration

For the FBK tools, in particular OCRA,nuXmv and xSAP, it has been realized a specific OSLC Provider based on the OSLC Automation domain. For test purposes only, a running service provider is available at the web site http://docker-es.fbk.eu:8080.

More specifically:

Table 7. Example of OSLC related web addresses

OSLC Service Catalogue	http://docker-es.fbk.eu:8080/oslc4j-registry/catalog
List of available Plans	http://docker-es.fbk.eu:8080/eu.fbk.tools.oslc.provider/services/autoPlans
List of Requests	http://docker-es.fbk.eu:8080/eu.fbk.tools.oslc.provider/services/autoRequests
List of Results	http://docker-es.fbk.eu:8080/eu.fbk.tools.oslc.provider/services/autoResults

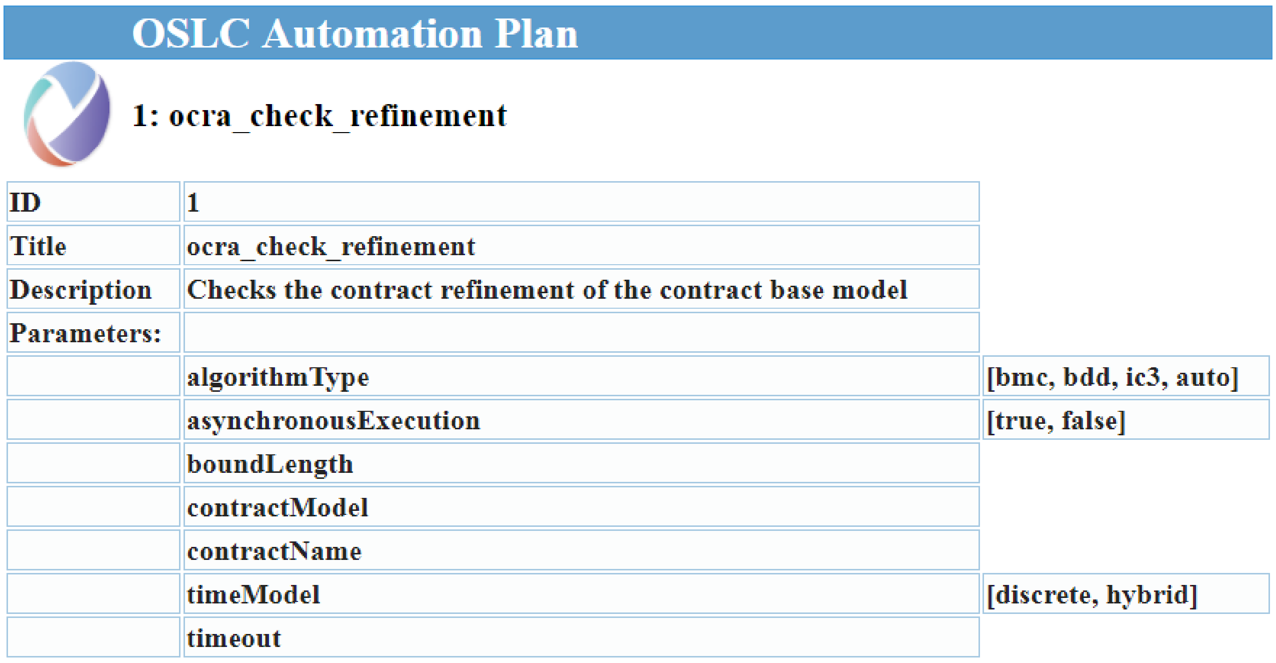
The FBK tool functionalities are made available by means of Automation Plans; the `title` attribute of the Automation Plan selects the tool functionality. At the current time, the available plans are the following:

Table 8. Examples of V&V related functionality and automation plans

FBK Tool Functionality	Automation Plan title
Check Contract refinement	**ocra_check_refinement**
Check Contract Implementation	**ocra_check_implementation**
Check Contract Validation Property	**ocra_check_validation_prop**
Compute Fault Tree	**ocra_compute_fault_tree**
Behaviour Model Check	**nuxmv_check_model**
Compute Fault Tree	**xsap_compute_fault_tree**
Compute FMEA Table	**xsap_compute_fmea_table**
Extend Model	**xsap_extend_model**

So if one intends to perform for example the check of a model based on contracts, he should select the Plan that has the `<dcterms:title>` attribute value set to **`ocra_check_refinement`**:

All the parameters of the Plan are returned by the service provider sending the request on the Plan instance. For example, if the ID of the plan is 1 (see the catalogue for the whole ID list), the HTML response to the request http://docker-es.fbk.eu:8080/eu.fbk.tools.oslc.provider/services/autoPlans/1 will be as shown in Figure 29.



OSLC Automation Plan

1: ocra_check_refinement

ID	1	
Title	ocra_check_refinement	
Description	Checks the contract refinement of the contract base model	
Parameters:		
	algorithmType	[bmc, bdd, ic3, auto]
	asynchronousExecution	[true, false]
	boundLength	
	contractModel	
	contractName	
	timeModel	[discrete, hybrid]
	timeout	

Figure 29. OSLC Automation Plan

So, the steps to follow for executing a given function of the FBK tools are:

1. Find in the catalogue the URI of the Automation Plan that maps the tool function.
2. Create an Automation Request that links such Plan, set the input parameters (see the Plan for their definitions) and submit the Request to the service provider.
   Once the request has been submitted, its status can be "`in progress`" or "`completed`" depending if the request is synchronous or asynchronous.
3. When the Request is `completed`, ask the Service Provider for the availability of the Automation Result that is linked to the Request.
   (See the Result attribute `<oslc_auto:producedByAutomationRequest>`).

3.7 Seamless Tracing via OSLC for Safety Case Fragments Generation

This section assumes the existence of OSLC adaptors for tools associated to the production of inter-related life-cycle artefacts. This assumption is necessary to guarantee the presence of a tool-chain composed of at least two tools. For instance, the presence of OSLC adaptors for DOORS and Simulink would enable the seamless tracing of artefacts related to requirements engineering (RM) and design (AM).

To enable the generation of safety case fragments arguing about traceable life-cycle data in compliance with standards, standard-compliant OSLC domains need to be introduced. To introduce a new domain, two options are at disposal, either a new domain is introduced from scratch or a new domain is obtained via extension of a pre-existing one. When a domain already offers interesting resources, the second option is recommended. In both cases, before applying OSLC best practices, guidelines are needed to proceed. In this section, a set of guidelines is presented. These guidelines should be applied systematically to each clause in order to identify and model the needed resources. These guidelines are (see Figure 30):

- Identify the work product types that are required and are expected to be compiled during the creation of a safety case. For each listed work product, a corresponding work product type is defined and modelled as a meta-class.

  Note that in the case of ISO 26262, since each of its clauses is structured in the same way (objectives, general, inputs for the clause, requirements and recommendation, and work products), the identification of the work product types is straightforward since the list of work products is clearly defined.
- Identify the text that describes relevant information for characterising the work products. Once the text is identified, meta-attributes and/or other types are added in order to fully characterise the work product types. The meta-attributes are used to describe the work product types. The definition of the meta-attributes is found by analysing the requirements and recommendations sub-clause.
- Identify the text that describes associations that inter-relate the work products. Once the text is identified, meta-associations can be added to inter-relate the work product types.

As previously mentioned, these methodological guidelines should be applied to each clause. By doing so, and by exploiting the information related to the expected input of each clause, it is possible to establish the meta-association that relates one domain (e.g., QM) with other domains (e.g., AM and RM). Thus, it is possible, domain after domain, to reproduce an OSLC-based representation of the V-model life-cycle artefacts.

As a result, a set of inter-related meta-classes representing the targeted standard-compliant resources is obtained. For sake of readability and communication, we suggest to first depict the domain as a class diagram in compliance with a UML profile for OSLC.

The depicted meta-model is given in a human-readable format, which can be easily discussed with a set of experts to get their approval. Then, we proceed with the manual translation of the UML-based representation into an OSLC-based domain, given as RDF-Schema. Once the schema is created, an instance can be instantiated, populated, and represented as an RDF-graph. It is worth to note that this translation could be automated as explored.

Once a standard-compliant OSLC-based domain of domains, given as RDF-Schema, is instantiated based on real product-related data, appropriate queries can be formulated to retrieve the necessary information to build safety cases fragments [7].

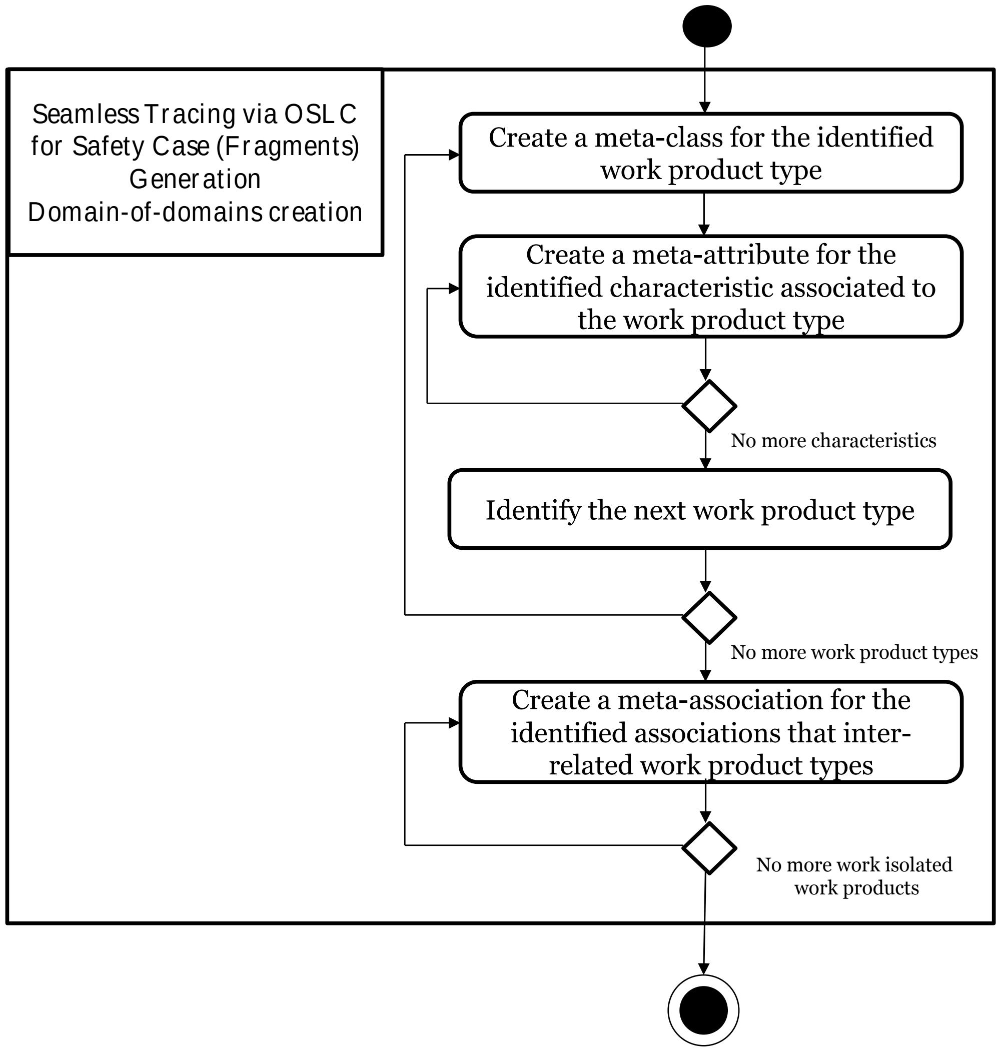

Seamless Tracing via OSLC for Safety Case (Fragments) Generation Domain-of-domains creation
Create a meta-class for the identified work product type
Create a meta-attribute for the identified characteristic associated to the work product type
No more characteristics
Identify the next work product type
No more work product types
Create a meta-association for the identified associations that inter-related work product types
No more work isolated work products

Figure 30. Domain-of-domains creation

3.8 Collaborative Editing

Implementation wise, a server component for collaborative editing is a simple storage for EMF resources with a REST API to control and modify them. There is no sophisticated revision control; this is out of scope of the prototype implementation. Anyway, the server is maintaining a full change history of all changes back to the original version. Clients may go back in time and retrieve all change sets individually to for example replay changes or catch up with changes from other users if they run out of sync.

The API is the same for all kind of EMF based editors (graphical GMF based editors as Papyrus, generate EEF forms but also generic property sheets generated by the EMF code generation), and does not depend on any metamodel. Note: Due to the technical architecture of AMASS and other GMF/EMF based tools, local modifications in the editor that initiated the modification request are typically not really deferred until the server has processed the request. Local modifications are instead applied immediately (and recorded by the transactional mechanism of the editor), and it is the result of this recording that is turned into a modification request and send to the server. The transaction commit is withheld until the modification acknowledgement is received from the server. If the modification received from the server does not correspond to the request, the transaction is aborted and rolled back. The editor may also play it "optimistic" to speed up performance and not wait for any result from the server but instead continue. Instead the editor has to reverse modifications later in case the deferred incoming response notifies a problem or conflict on the server side.

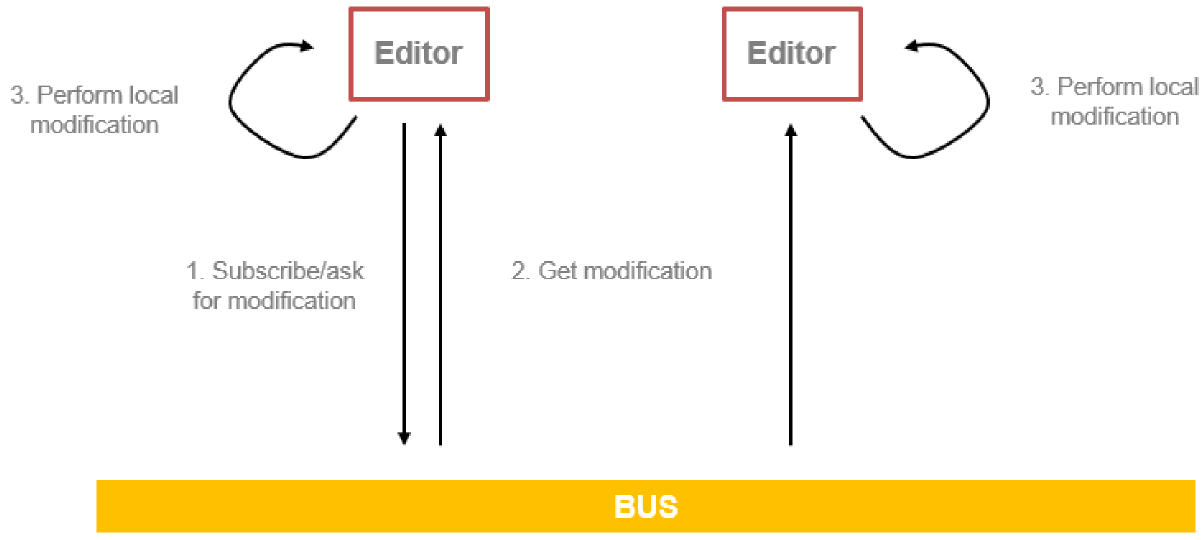

Editor
Editor
3. Perform local modification
3. Perform local modification
1. Subscribe/ask for modification
2. Get modification
BUS

Figure 31. Tool setup and workflow for collaborative editing

GMF and other EMF editors that support undo/redo have a so called "Transaction Domain" in place including a "Command Stack" and a "Change Recorder" (see open EMF and GMF sources and documentation). The transaction lifecycle and the change recorder are utilized on client side to both collect and send change sets to the server but also apply received changes from the server. After a command (or several commands in a compound command) were executed - so changes were made in the client and the transaction is about to close -, all changes were recorded and are available as "Change Descriptions". These descriptions are transformed into applicable EMF commands, marshalled to JSON, send over the bus and applied to the server. The same commands are then send to all other clients, unmarshalled there, converted back to EMF commands and applied normally to the editing domain (put on the command stack). Technically there is no difference between a command that was executed by a user interaction and a command received from the server.

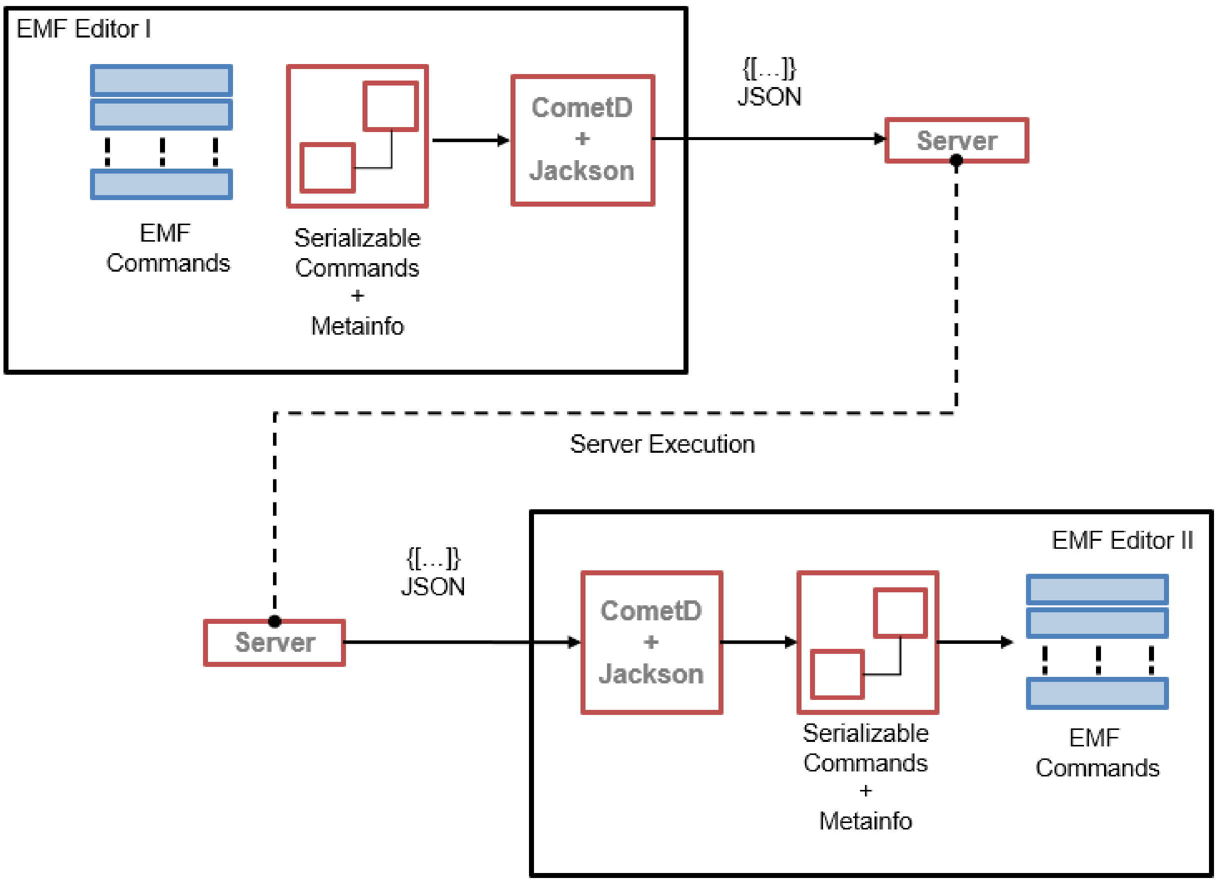


Figure 32. From change in one client, over collaborative server to change in another client

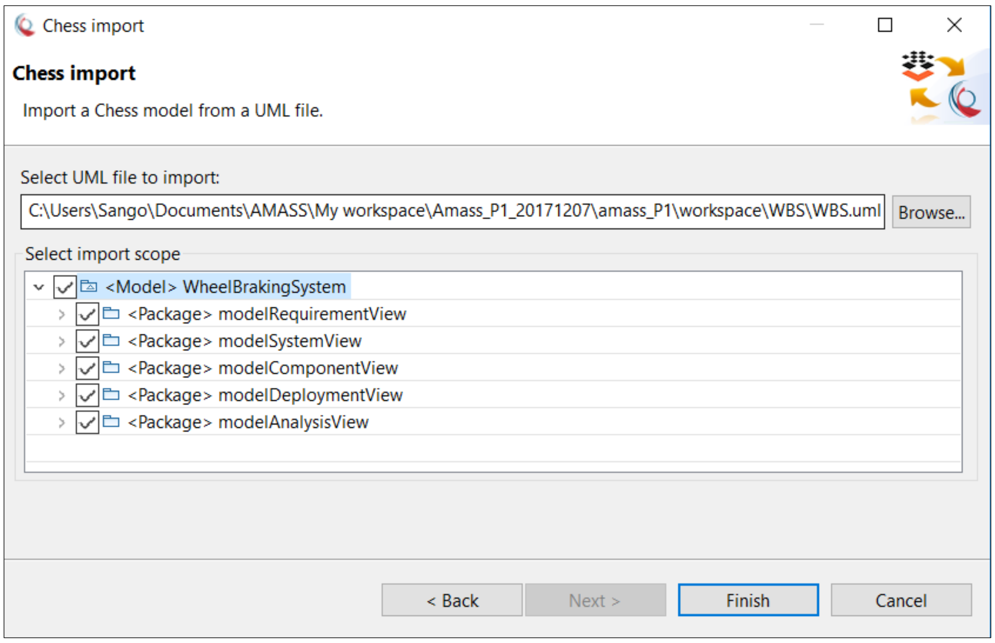
As mentioned, this approach can be easily integrated into any EMF/GMF editor. As a proof of concept, it has been integrated into the Papyrus editor, or more precisely, it has been integrated into the ANSYS SCADE Architect tool, which is based on Papyrus. The tool uploads a new Papyrus model from - or downloads an existing Papyrus model to - the server (via the API) - and hooks into the EMF transactions mechanism to send/receive commands. The same was applied to the medini FTA editor as a proof of concept to be applicable to different types of editors.

As mentioned above, the used technology stack is a commercial product. That means, the server and the Java glue code that makes the REST API and the BUS accessible in Java is closed source. However, the used libraries and underlying technologies as CometD (for the mentioned BUS and libraries as Jackson for REST access) are freely available and even open source. The approach and the technology is easily adoptable by any tool vendor that would like to attach its editor on the same platform. Though it is not working out of the box yet, some code changes are required. A solution that works generically for any EMF editor without touching the code is still under investigation.

3.9 Safety/Cyber Architect Tools Integration

The Safety Architect and Cyber Architect tools are integrated with the AMASS platform as external tools to provide support for safety and security co-analysis. The integration plugins for the transformation from CHESS to Safety Architect are described in the deliverable D5.6 [14]. To use the Safety Architect and Cyber Architect tools integration, the user must follow the following steps:

- **Step 1:** Import a CHESS model from a UML file, as illustrated in Figure 33.


Chess import

Chess import

Import a Chess model from a UML file.

Select UML file to import:

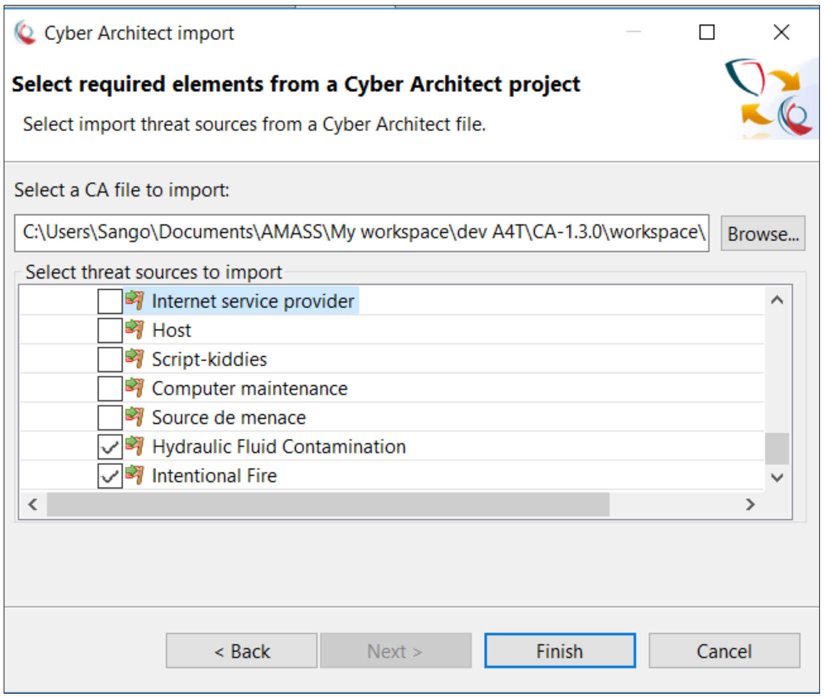
C:\Users\Sango\Documents\AMASS\My workspace\Amass_P1_20171207\amass_P1\workspace\WBS\WBS.uml

Select import scope

- ✓ <Model> WheelBrakingSystem
  - ✓ <Package> modelRequirementView
  - ✓ <Package> modelSystemView
  - ✓ <Package> modelComponentView
  - ✓ <Package> modelDeploymentView
  - ✓ <Package> modelAnalysisView

Figure 33. Import from CHESS tool to Safety Architect tool

- **Step 2:** Import data from Cyber Architect file, as illustrated in Figure 34.


Cyber Architect import

Select required elements from a Cyber Architect project

Select import threat sources from a Cyber Architect file.

Select a CA file to import:

C:\Users\Sango\Documents\AMASS\My workspace\dev A4T\CA-1.3.0\workspace\

Select threat sources to import

- ☐ Internet service provider
- ☐ Host
- ☐ Script-kiddies
- ☐ Computer maintenance
- ☐ Source de menace
- ✓ Hydraulic Fluid Contamination
- ✓ Intentional Fire

Figure 34. Import Data from Cyber Architect tool to Safety Architect tool

- **Step 3:** Assurance Engineers can activate the Security Viewpoint in Safety Architect tool for Safety and Security Co-analysis. The activation of Safety & Security viewpoint in Safety Architect allows the annotation of input and output ports of system components with Security Thread Modes (e.g., hydraulic fluid contamination or intentional fire) imported in previous step. The co-analysis is realized thanks to these threat modes, failure modes (internal failure, erroneous) and logical gates, as illustrated in Figure 35.

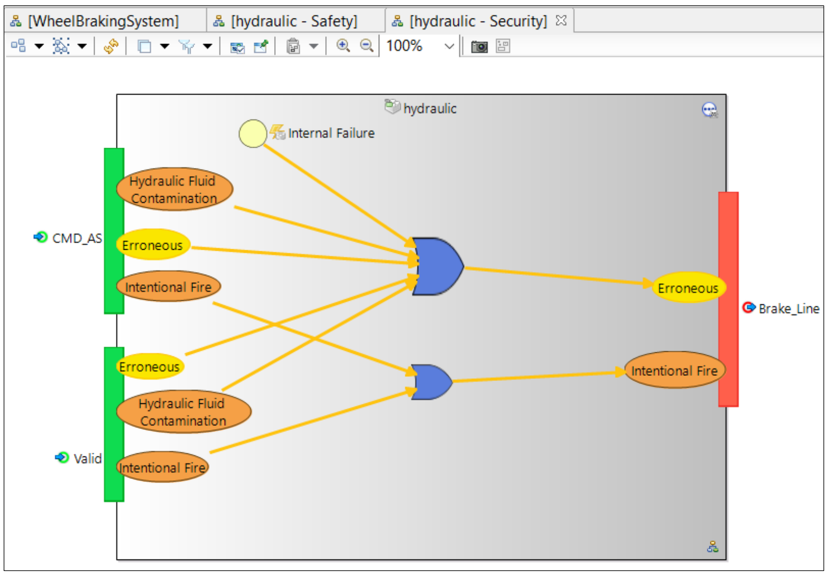
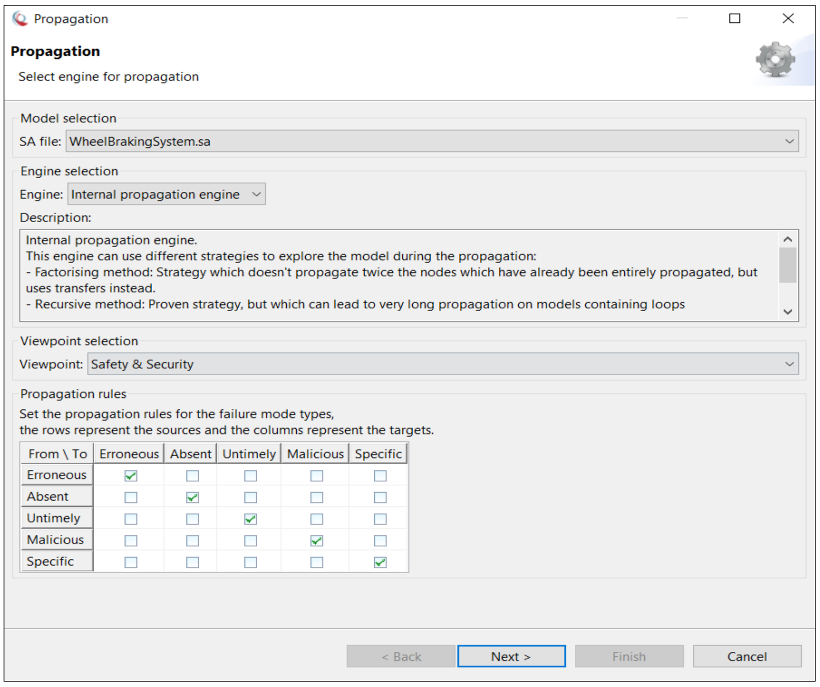


Figure 35. Safety & Security Viewpoint in Safety Architect tool

- **Step 4:** Assurance Engineers can generate the Safety & Security artefacts (e.g., Faults and Attacks Propagation Tree) thanks to the previous Safety & Security co-analysis and the Safety Architect propagation engine with the selection of "Safety & Security" viewpoint, as illustrated in Figure 36.



Propagation

Propagation

Select engine for propagation

Model selection

SA file: WheelBrakingSystem.sa

Engine selection

Engine: Internal propagation engine

Description:

Internal propagation engine.
This engine can use different strategies to explore the model during the propagation:
- Factorising method: Strategy which doesn't propagate twice the nodes which have already been entirely propagated, but uses transfers instead.
- Recursive method: Proven strategy, but which can lead to very long propagation on models containing loops

Viewpoint selection

Viewpoint: Safety & Security

Propagation rules

Set the propagation rules for the failure mode types,
the rows represent the sources and the columns represent the targets.

From \ To	Erroneous	Absent	Untimely	Malicious	Specific
Erroneous	✓	☐	☐	☐	☐
Absent	☐	✓	☐	☐	☐
Untimely	☐	☐	✓	☐	☐
Malicious	☐	☐	☐	✓	☐
Specific	☐	☐	☐	☐	✓

Figure 36. Safety & Security Viewpoint Selection in Safety Architect Tool

The failures and threats propagation tree generated in Safety Architect is shown in Figure 37.

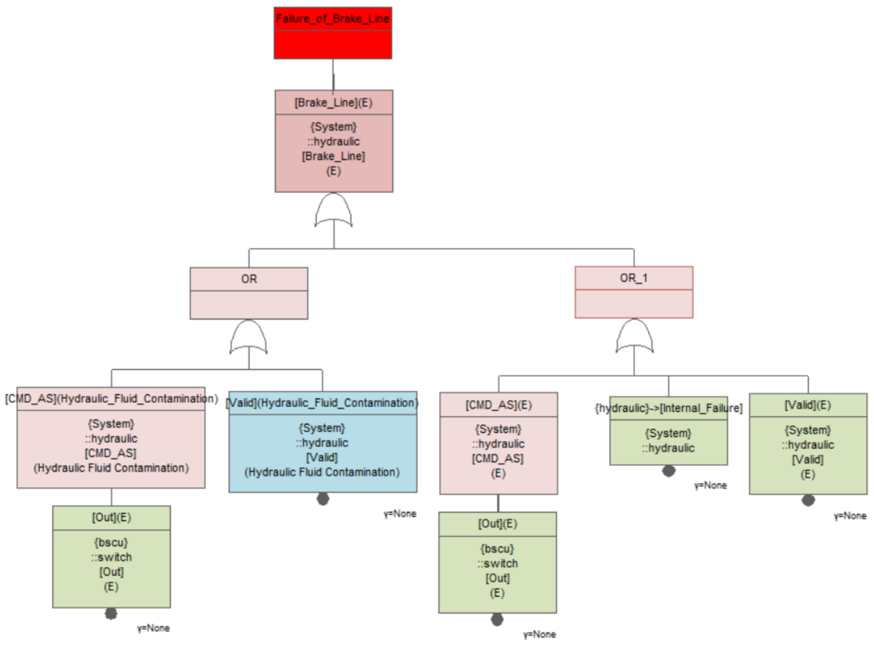
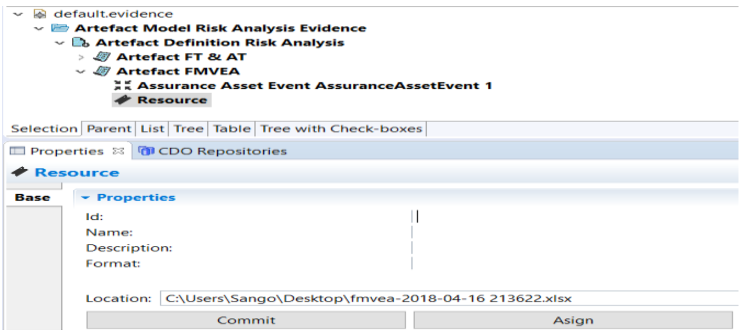

Failure_of_Brake_Line
[Brake_Line](E)
{System} ::hydraulic [Brake_Line] (E)
OR
OR_1
[CMD_AS](Hydraulic_Fluid_Contamination)
{System} ::hydraulic [CMD_AS] (Hydraulic Fluid Contamination)
[Valid](Hydraulic_Fluid_Contamination)
{System} ::hydraulic [Valid] (Hydraulic Fluid Contamination)
[CMD_AS](E)
{System} ::hydraulic [CMD_AS] (E)
{hydraulic}->[Internal_Failure]
{System} ::hydraulic
[Valid](E)
{System} ::hydraulic [Valid] (E)
[Out](E)
{bscu} ::switch [Out] (E)
[Out](E)
{bscu} ::switch [Out] (E)
γ=None
γ=None
γ=None
γ=None
γ=None

Figure 37. Propagation Tree in Safety Architect Tool

The integration of AMASS platform tools (CHESS and OpenCert) with safety/security analysis tools (Safety Architect and Cyber Architect) also supports:

- Evidence resource specification, because an assurance engineer can indicate in OpenCert the location of the evidence resource, such as Fault/Attack Trees or FMEA/FMVEA tables, generated in Safety Architect tool, shown in Figure 38:



default.evidence
Artefact Model Risk Analysis Evidence
Artefact Definition Risk Analysis
Artefact FT & AT
Artefact FMVEA
Assurance Asset Event AssuranceAssetEvent 1
Resource

Selection | Parent | List | Tree | Table | Tree with Check-boxes

Properties | CDO Repositories

Resource

Base | Properties

Id:
Name:
Description:
Format:

Location: C:\Users\Sango\Desktop\fmvea-2018-04-16 213622.xlsx

Commit | Asign

Figure 38. Evidence resource location in OpenCert

- Visualization of evidence chains, because the fault/attack trees generated in Safety Architect can be back propagate in CHESS model and displayed as chains of evidence as indicated in Figure 39:

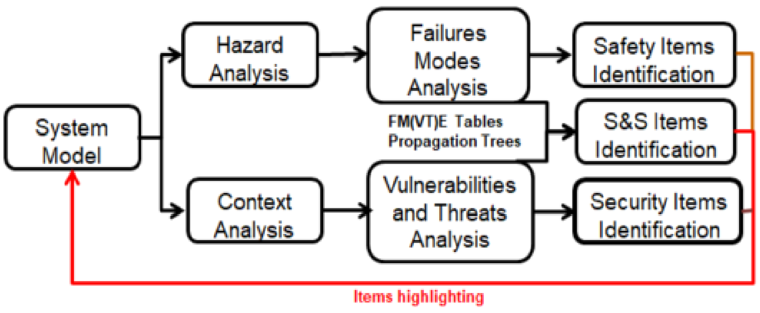

System Model
Hazard Analysis
Failures Modes Analysis
Safety Items Identification
FM(VT)E Tables Propagation Trees
S&S Items Identification
Context Analysis
Vulnerabilities and Threats Analysis
Security Items Identification
Items highlighting

Figure 39. A Process for system safety and security co-analysis

3.10 Data and Security Management

In a company where the AMASS Platform is deployed, there should be a general AMASS Platform administrator. This role should be assigned to someone from the own IT services staff. This person will be responsible of managing the users with access to the AMASS Platform.

It is important that passwords are strong enough, for that it is highly recommendable that passwords contain at least one capital letter, one number and one special character, being no less than 8 characters long.

As mentioned in D2.4 [19], the AMASS Platform can be used by different stakeholders. The data and security management will mainly be used by the manufacturer group.

The **Project Manager** is the role that works on a compliance and assurance-based project where a product (system or component) needs to be assessed as acceptably dependable (safety, security or other dependability properties). The Project Manager will use the AMASS Tool Platform to check the status of the project's goals within the planned budget, time, and resources. The Project Manager (administrator role) can access the data with read permissions, however, will in principle not be able to modify any data.

The **Assurance Manager** is responsible to show compliance with a particular standard by means of an assurance case. The Assurance Manager will use the ARTA platform to plan, structure, view, review and assess the system structure and arguments or modules, sometimes by composing pre-existing arguments, and reusing arguments and evidence relating to reusable components. In the same way that an Assurance Engineer can be split in two groups (i.e. Safety and Security engineer), an Assurance Manager can be subdivided into Safety and Security manager. The Assurance Manager (administrator role) can access the data with read and write permissions.

The **Assurance Engineer** is the role responsible for executing the different V&V and assurance activities e.g., create and/or collect the evidence to demonstrate that the product is acceptable safe/secure. An assurance engineer can be split into Safety and Security engineer roles. The Assurance Engineer (User Role) can access the data with read and write permissions.

The **Internal Assessor** is responsible for assessing the adequacy of the evidence and assurance 'package', in terms of demonstrating the safety/security of the system under consideration. The Internal Assessor (General Role) can access all the data with read permissions.

As mentioned in D2.4 [19], another stakeholder is the authority which can be the National Safety/Security Authority, European Safety/Security Authority or the Regulator. The authority, and more specifically the **Assurance Assessor,** is responsible for assessing the adequacy of the evidence and assurance 'package' provided by the manufacturers, in terms of justifying the safety/security of the system or component under consideration. The Safety/Security assessor will use the AMASS tools to view workflows, arguments, compliance checklists and evidence artefacts related to the system or component. The Assurance Assessor (Reader Role) can only access and read the data stored under one specific assurance project.

4. Conclusions

It is not realistic to expect that a single tool will cover all needs of all CPS developers all the time. Instead, different tools may be used for certain purposes by various users. Such complex activities as CPS development typically require support from several tools. In order to keep the managed information (requirements, specifications, models, code, tests, traces, etc.) concise, non-redundant, and consistent, the employed tools should be able to communicate and understand each other. This deliverable presents the description of various approaches to seamless interoperability of relevant tools.

The main areas related to the interoperability are the representations of the shared data, the protocols for sending and receiving the data, and the ways of offering and invoking the services provided by the tools. These conceptual domains and related methodologies (e.g. OSLC with RDF and HTTP, Java plugins, and Capra) are covered by this document. The overview and insights into the existing approaches are needed by the implementers of the sender/receiver components and of the communication channels and by the modellers of the shared content. The process that can help to select the most appropriate approach for a given situation is presented in the form of an EPF process in the Appendix A. Methodological Guide for Seamless Interoperability – EPF Process Description.

The "seamless" aspect of the interoperability is stressed in order to minimize the burden imposed on the users of the interconnected development platform components. Seamless interoperability increases the usability of a cluster of cooperating tools, increases the understandability of the managed artefacts and their relationships by providing more perspectives on looking at the data, and it also prevents de-synchronization, loss or unnecessary re-entering of data.

Abbreviations and Definitions

Abbreviation	Explanation
ALM	Application Lifecycle Management
AM	Architecture Management
API	Application Programming Interface
ARTA	AMASS Reference Tool Architecture
CDO	Connected Data Objects
CHESS	Composition with Guarantees for High-integrity Embedded Software Components Assembly
CM	Change Management
COM	Component Object Model
CPS	Cyber Physical System
DNG	Diversity Network Group
DOORS	Dynamic Object-Oriented Requirements System
EEF	Extended Editing Framework
EMF	Eclipse Modelling Framework
EMS	Estimation and Measurement
EPF	Eclipse Process Framework
FMEA	Failure Mode and Effects Analysis
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FMVEA	Failure Modes, Vulnerabilities and Effect Analysis
FTA	Fault Tree Analysis
GMF	Graphical Modelling Framework
KM	Knowledge Management
KPI	Key Performance Indicator
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
INCOSE	International Council on Systems Engineering
ISO	International Organization for Standardization
IT	Information Technology
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LTL	Linear Temporal Logic
MBSE	Model-Based Systems Engineering
OCRA	Othello Contracts Refinement Analysis
OMG	Object Management Group
OPENCOSS	Open Platform for EvolutioNary Certification Of Safety-critical Systems
OSLC	Open Services for Lifecycle Collaboration
PBS	Product Breakdown Structure
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
OWL	Ontology Web Language
PBS	Product Breakdown Structure

PLM	Product Lifecycle Management
QM	Quality Management
RAT	Requirements Authoring Tool
RDF	Resource Description Framework
ReqIF	Requirements Interchange Format
RDFS	Resource Description Framework
REST	Representational State Transfer
RM	Requirements Management
RMS	Rights Management services
RMT	Requirements Management Tool
RQA	Requirements Quality Analyser
RQS	Requirements Quality Suite
RSA	Rational Software Architect
RTE	Real Time Edition
RTF	Rich Text Format
SKOS	Simple Knowledge Organization System
SHACL	Shapes Constraint Language
SKR	Software Knowledge Repository
SMT	Satisfiability Modulo Theory
SW	Software
SPARQL	SPARQL Protocol And RDF Query Language
SRL	System Representation Language
SysML	System Modelling Language
TRC	The REUSE Company
UML	Unified Modelling Language
URI	Uniform Resource Identifier
V&V	Verification & Validation
W3C	World Wide Web Consortium
XAT	Requirements Authoring Resident Process
XML	Extensible Markup Language
xSAP	eXtended Safety Assessment Platform

References

[1] Capra tool: https://projects.eclipse.org/proposals/capra

[2] de la Vara, J.L., Borg, M., Wnuk, K., Moonen, L.: An Industrial Survey on Safety Evidence Change Impact Analysis Practice. IEEE Transactions on Software Engineering 42(12): 1095 – 1117 (2016)

[3] Gallina, B., Nyberg, M.: Reconciling the ISO 26262-compliant and the Agile Documentation Management in the Swedish Context. Critical Automotive applications: Robustness & Safety (CARS) (2015)

[4] Gallina, B., Castellanos Ardila, J. P., Nyberg, M.: Towards Shaping ISO 26262-compliant Resources for OSLC-based Safety Case Creation. Critical Automotive applications: Robustness & Safety (CARS) (2016)

[5] Gallina, B., Padira, K., Nyberg, M.: Towards an ISO 26262-compliant OSLC-based tool chain enabling continuous self-assessment. 10th International Conference on the Quality of Information and Communications Technology- Track: Quality Aspects in Safety Critical Systems (QUATIC) (2016)

[6] Gallina, B.: Towards an ISO 26262-compliant OSLC-based Tool Chain Enabling Continuous Self-assessment. http://safety.addalot.se/upload/2017/2-3-1%20Gallina.pdf

[7] Gallina, B., Nyberg, M.: Pioneering the Creation of ISO 26262-compliant OSLC-based Safety Cases. WoSoCer 2017

[8] AMASS D2.1 Business cases and high-level requirements, February 2017

[9] AMASS D2.5 AMASS user guidance and methodological framework, October 2018

[10] AMASS D5.1 Baseline requirements for seamless interoperability, September 2016

[11] AMASS D5.3 Design of the AMASS tools and methods for seamless interoperability (b), September 2018

[12] AMASS D5.4 Prototype for seamless interoperability (a), March 2017

[13] AMASS D5.5 Prototype for seamless interoperability (b), November 2017

[14] AMASS D5.6 Prototype for seamless interoperability (c), September 2018

[15] OPENCOSS project. 2015. http://www.opencoss-project.eu/

[16] SafeCer Project. 2015. http://cordis.europa.eu/project/rcn/103721_en.html and http://cordis.europa.eu/project/rcn/105610_en.html

[17] OSLC community. 2017. https://open-services.net/

[18] AMASS D1.1 Case studies description and business impact, November 2016

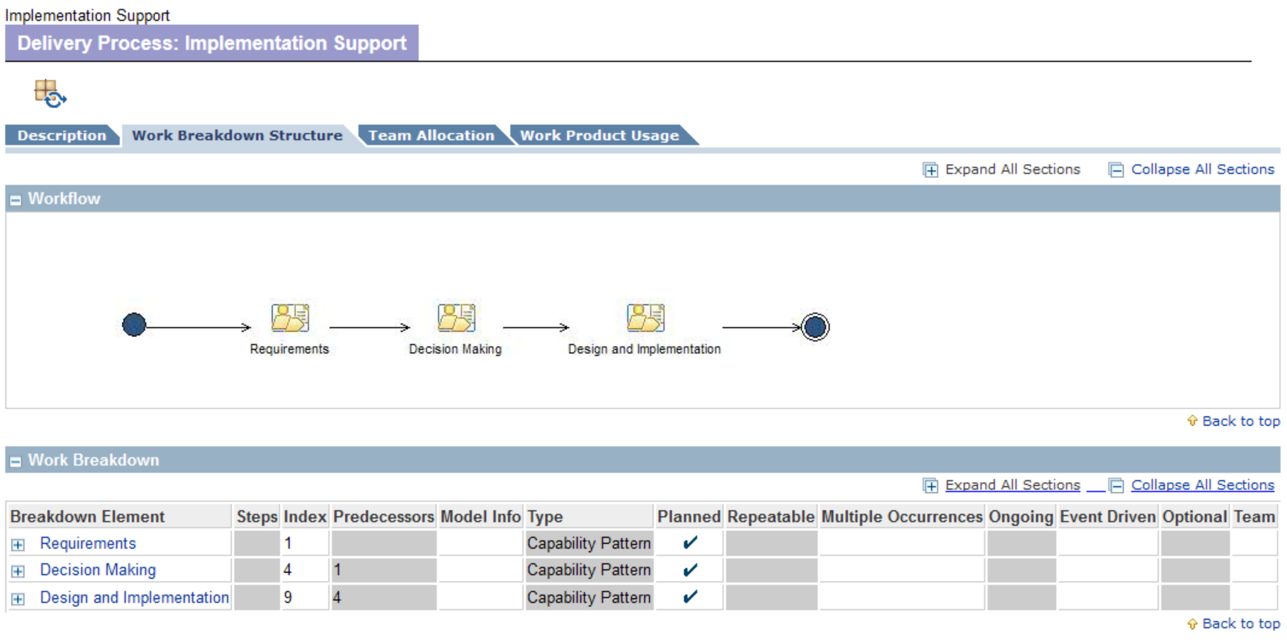
[19] AMASS D2.4 - AMASS reference architecture (c), May 2018

[20] https://www.reusecompany.com/knowledgemanager

Appendix A. Methodological Guide for Seamless Interoperability – EPF Process Description

This appendix summarises the process in the form provided by the EPF Composer. This form is highly structured and optimized in the sense, that it instantiates an elaborated process meta-model, minimizes duplicated data, and generates several useful views of the same information in various contexts. The work breakdown structure is visualised in a type of activity/workflow diagram, which increases the understandability of the process description by providing a simple overview of the process components and their interdependence.

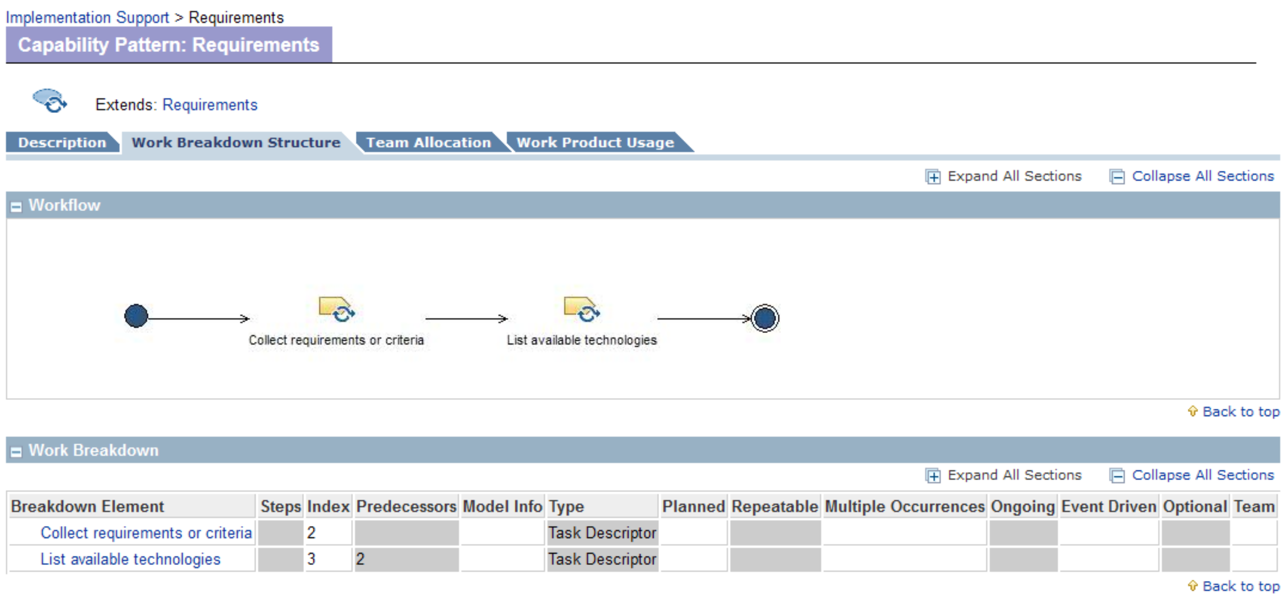
The top-level overview of the process that helps the implementors of the seamless interoperability to choose the appropriate technology is depicted in the Figure 40. This process is further decomposed to the activities shown in the Figure 41, the Figure 44, and the Figure 55.



Breakdown Element	Steps	Index	Predecessors	Model Info	Type	Planned	Repeatable	Multiple Occurrences	Ongoing	Event Driven	Optional	Team
Requirements		1			Capability Pattern	✔						
Decision Making		4	1		Capability Pattern	✔						
Design and Implementation		9	4		Capability Pattern	✔						

Figure 40. Top-level overview of the process

The Figure 41 captures the initial stage of an integration of a new tool, namely the identification of the current context –what is required and what is available.


Implementation Support > Requirements

Capability Pattern: Requirements

Extends: Requirements

Description | Work Breakdown Structure | Team Allocation | Work Product Usage

Expand All Sections Collapse All Sections

Workflow

Back to top

Work Breakdown

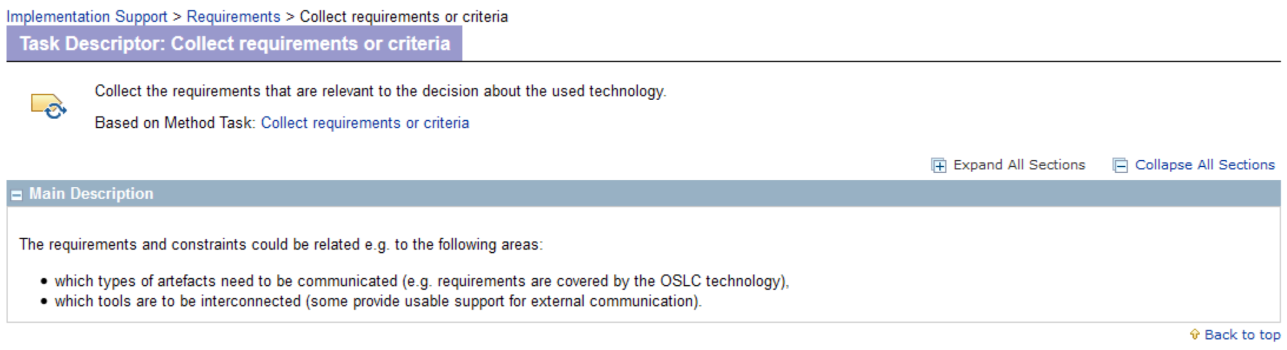
Expand All Sections Collapse All Sections

Breakdown Element	Steps	Index	Predecessors	Model Info	Type	Planned	Repeatable	Multiple Occurrences	Ongoing	Event Driven	Optional	Team
Collect requirements or criteria		2			Task Descriptor							
List available technologies		3	2		Task Descriptor							

Back to top

Figure 41. Activity diagram for the first stage of the process

The Figure 42 provides details on the activity introduced in the Figure 41.


Implementation Support > Requirements > Collect requirements or criteria

Task Descriptor: Collect requirements or criteria

Collect the requirements that are relevant to the decision about the used technology.

Based on Method Task: Collect requirements or criteria

Expand All Sections Collapse All Sections

Main Description

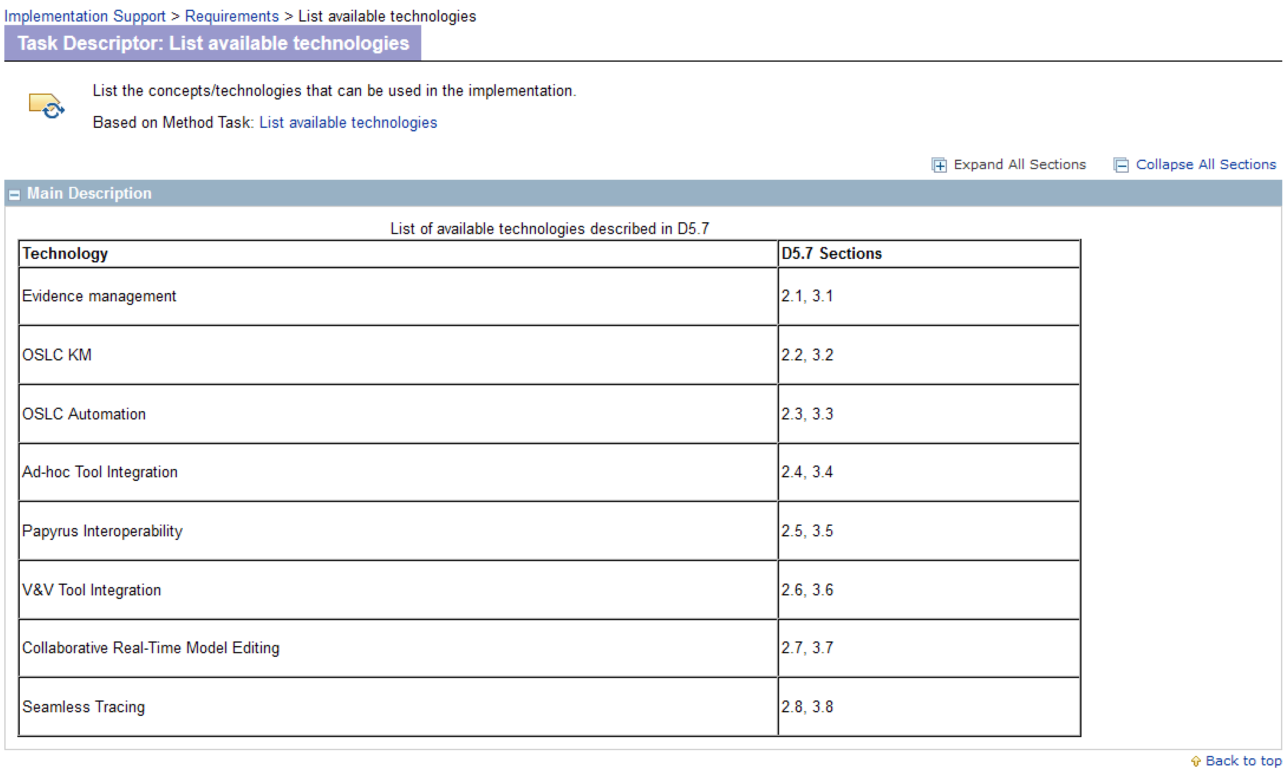
The requirements and constraints could be related e.g. to the following areas:

- which types of artefacts need to be communicated (e.g. requirements are covered by the OSLC technology),
- which tools are to be interconnected (some provide usable support for external communication).

Back to top

Figure 42. Description of the task *Collect requirements or criteria*

The Figure 43 provides details on the activity introduced in the Figure 41, including how the sections of this deliverable are related to individual technologies.


Implementation Support > Requirements > List available technologies

Task Descriptor: List available technologies

List the concepts/technologies that can be used in the implementation.

Based on Method Task: List available technologies

Expand All Sections Collapse All Sections

Main Description

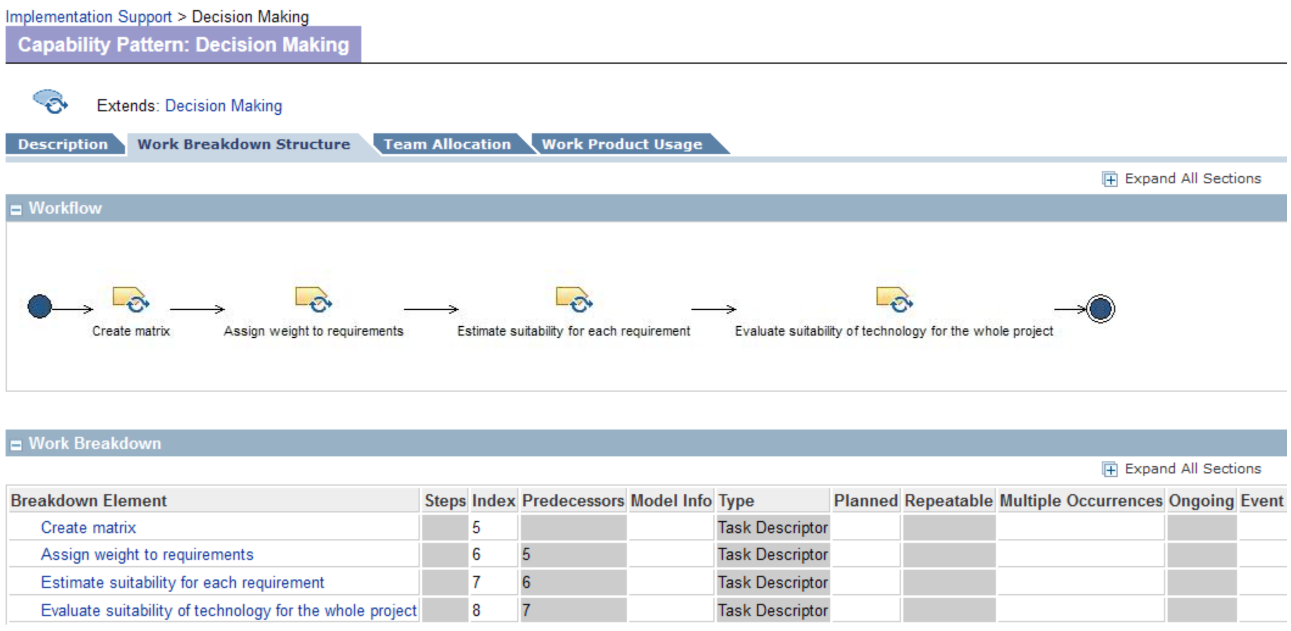
List of available technologies described in D5.7

Technology	D5.7 Sections
Evidence management	2.1, 3.1
OSLC KM	2.2, 3.2
OSLC Automation	2.3, 3.3
Ad-hoc Tool Integration	2.4, 3.4
Papyrus Interoperability	2.5, 3.5
V&V Tool Integration	2.6, 3.6
Collaborative Real-Time Model Editing	2.7, 3.7
Seamless Tracing	2.8, 3.8

Back to top

Figure 43. Description of the task *List available technologies*

The Figure 44 elaborates the decision making for the selection of an appropriate implementation technology. The figure contains one part of the decomposition of the overall process presented in the Figure 40.


Implementation Support > Decision Making

Capability Pattern: Decision Making

Extends: Decision Making

Description | Work Breakdown Structure | Team Allocation | Work Product Usage

Expand All Sections

Workflow

Create matrix
Assign weight to requirements
Estimate suitability for each requirement
Evaluate suitability of technology for the whole project

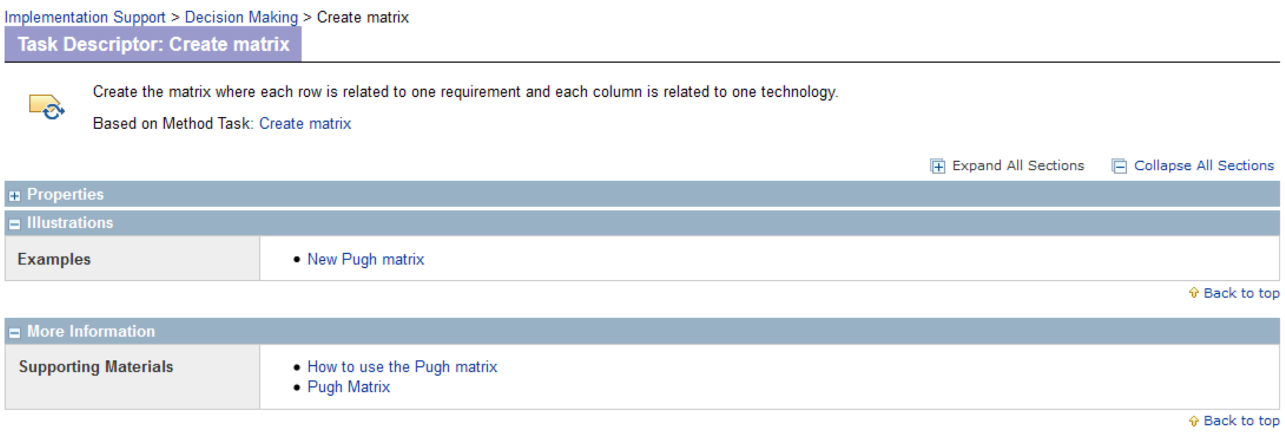
Work Breakdown

Expand All Sections

Breakdown Element	Steps	Index	Predecessors	Model Info	Type	Planned	Repeatable	Multiple Occurrences	Ongoing	Event
Create matrix		5			Task Descriptor					
Assign weight to requirements		6	5		Task Descriptor					
Estimate suitability for each requirement		7	6		Task Descriptor					
Evaluate suitability of technology for the whole project		8	7		Task Descriptor					

Figure 44. Activity diagram of *Decision making*

The Figure 45 describes one of the activities depicted in the Figure 44.


Implementation Support > Decision Making > Create matrix

Task Descriptor: Create matrix

Create the matrix where each row is related to one requirement and each column is related to one technology.

Based on Method Task: Create matrix

Expand All Sections Collapse All Sections

Properties

Illustrations

Examples	• New Pugh matrix

Back to top

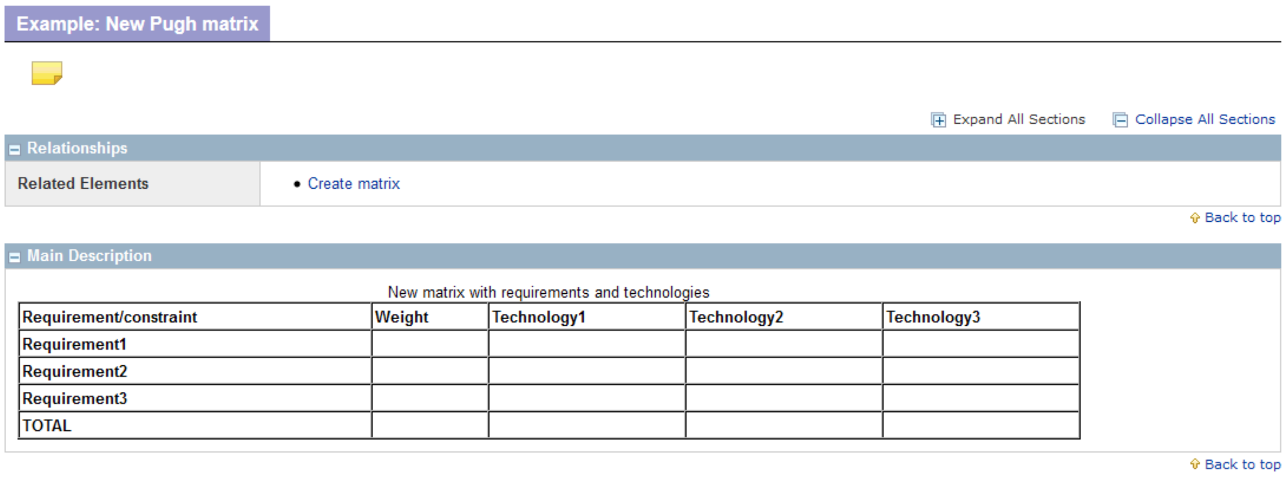
More Information

Supporting Materials	• How to use the Pugh matrix • Pugh Matrix

Back to top

Figure 45. Description of task *Create matrix*

The Figure 46 shows an example of the result of the task described in the Figure 45.


Example: New Pugh matrix

Expand All Sections Collapse All Sections

Relationships

Related Elements	• Create matrix

Back to top

Main Description

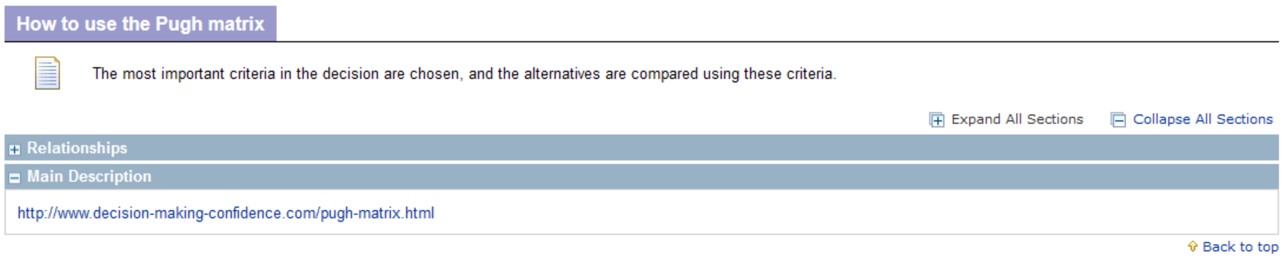
New matrix with requirements and technologies

Requirement/constraint	Weight	Technology1	Technology2	Technology3
Requirement1				
Requirement2				
Requirement3				
TOTAL				

Back to top

Figure 46. Example of new Pugh matrix

The Figure 47 and Figure 48 contain the references to supporting materials related to the decision process.


How to use the Pugh matrix

The most important criteria in the decision are chosen, and the alternatives are compared using these criteria.

Expand All Sections Collapse All Sections

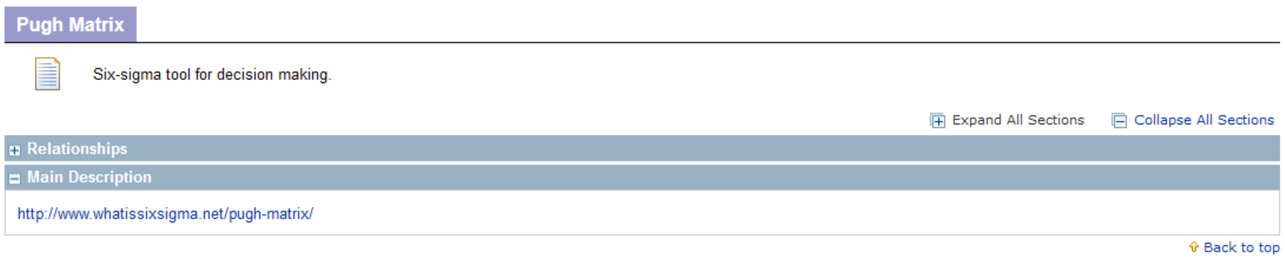
Relationships

Main Description

http://www.decision-making-confidence.com/pugh-matrix.html

Back to top

Figure 47. Reference to the supporting material *How to use the Pugh matrix*


Pugh Matrix

Six-sigma tool for decision making.

Expand All Sections Collapse All Sections

Relationships

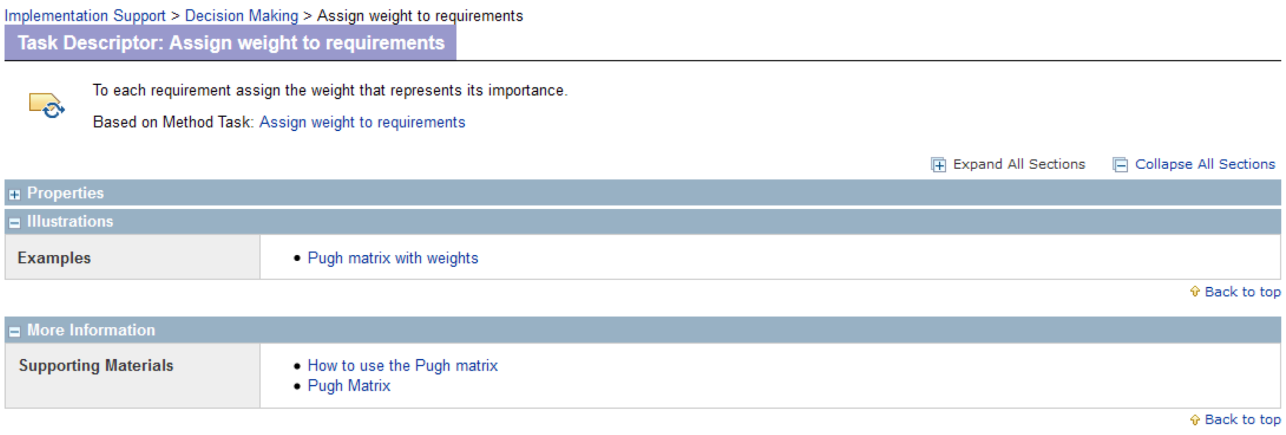
Main Description

http://www.whatissixsigma.net/pugh-matrix/

Back to top

Figure 48. Reference to the supporting material *Pugh matrix*

The Figure 49 further describes the details of the decision-making process introduced at its top-level in the Figure 44.


Implementation Support > Decision Making > Assign weight to requirements

Task Descriptor: Assign weight to requirements

To each requirement assign the weight that represents its importance.

Based on Method Task: Assign weight to requirements

Expand All Sections Collapse All Sections

Properties

Illustrations

Examples	• Pugh matrix with weights

Back to top

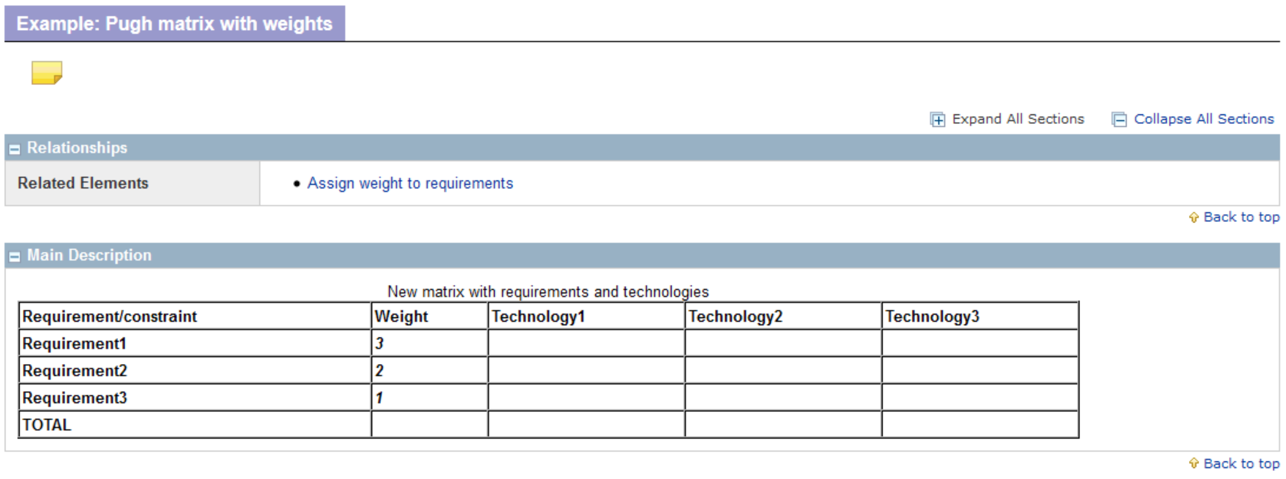
More Information

Supporting Materials	• How to use the Pugh matrix • Pugh Matrix

Back to top

Figure 49. Description of the task *Assign weight to requirements*

The Figure 50 is a follow-up to the example given in the Figure 46 after performing the task described in the Figure 49.


Example: Pugh matrix with weights

Expand All Sections Collapse All Sections

Relationships

Related Elements	• Assign weight to requirements

Back to top

Main Description

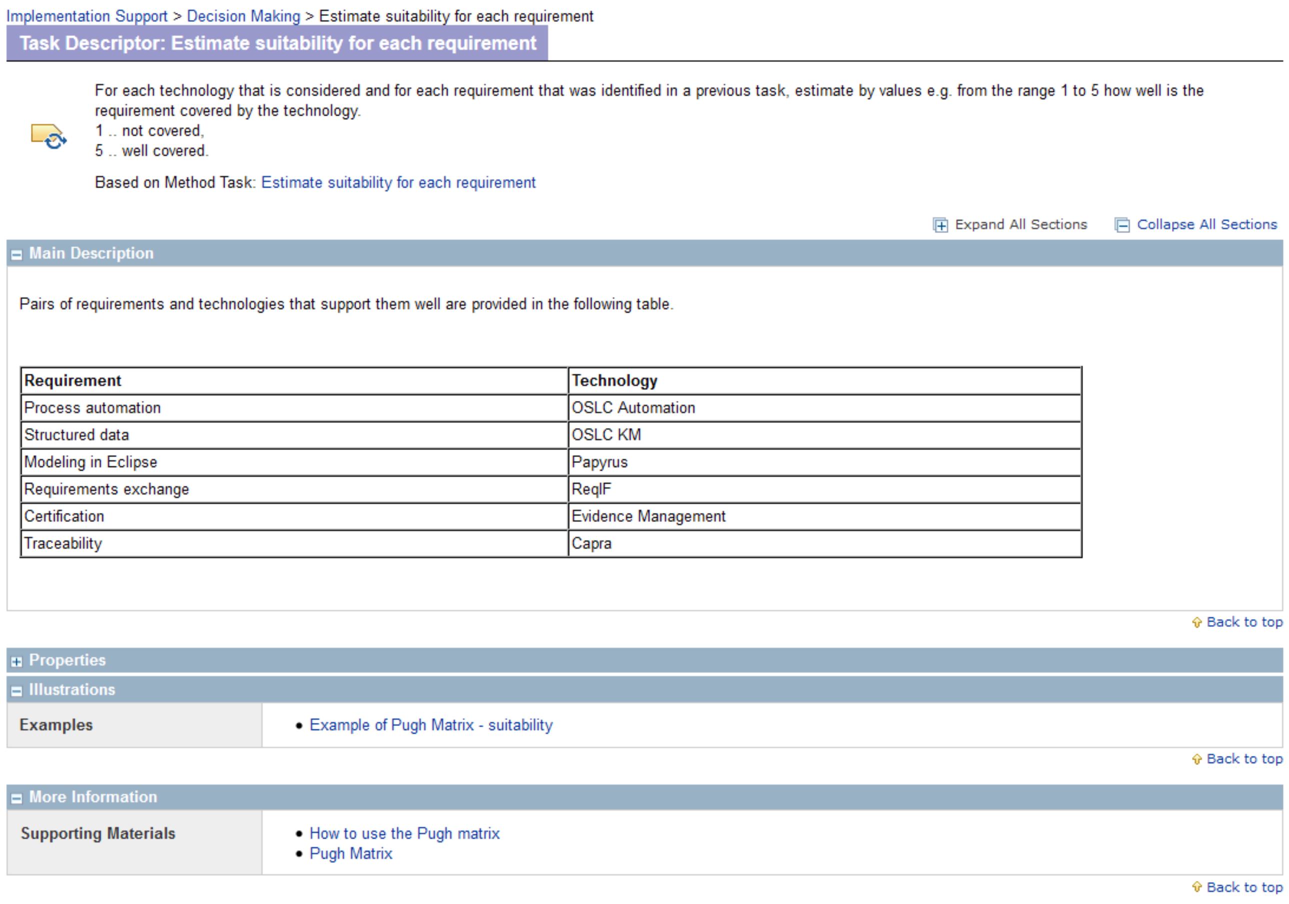
New matrix with requirements and technologies

Requirement/constraint	Weight	Technology1	Technology2	Technology3
Requirement1	3			
Requirement2	2			
Requirement3	1			
TOTAL				

Back to top

Figure 50. Example of Pugh matrix with weights

Another task which is part of the decision making (shown in the Figure 44) is described in the Figure 51.



Implementation Support > Decision Making > Estimate suitability for each requirement

Task Descriptor: Estimate suitability for each requirement

For each technology that is considered and for each requirement that was identified in a previous task, estimate by values e.g. from the range 1 to 5 how well is the requirement covered by the technology.
1 .. not covered,
5 .. well covered.

Based on Method Task: Estimate suitability for each requirement

Expand All Sections Collapse All Sections

Main Description

Pairs of requirements and technologies that support them well are provided in the following table.

Requirement	Technology
Process automation	OSLC Automation
Structured data	OSLC KM
Modeling in Eclipse	Papyrus
Requirements exchange	ReqIF
Certification	Evidence Management
Traceability	Capra

Back to top

Properties

Illustrations

Examples	• Example of Pugh Matrix - suitability

Back to top

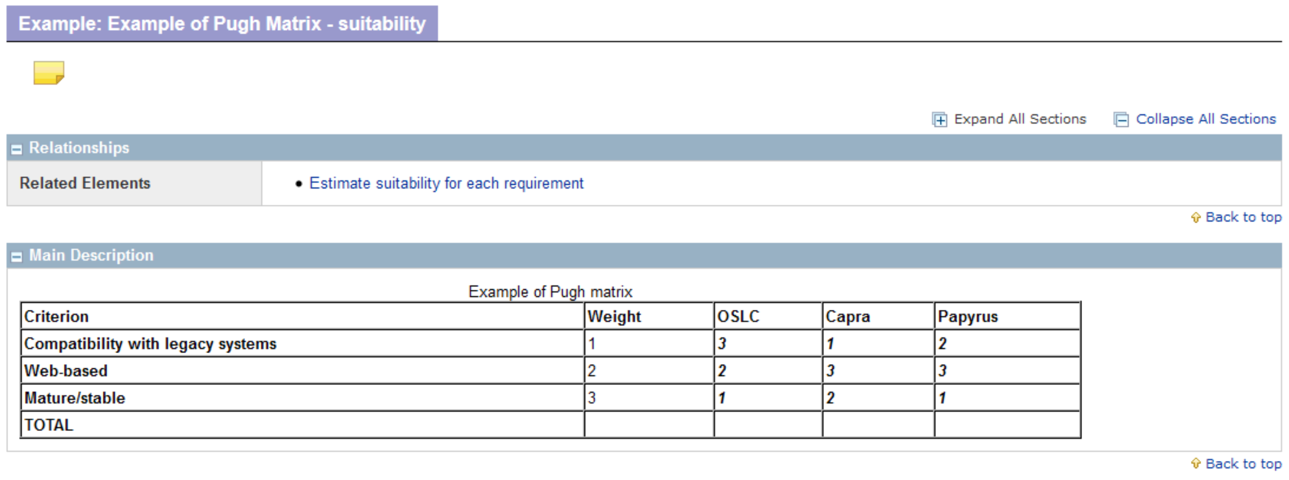
More Information

Supporting Materials	• How to use the Pugh matrix • Pugh Matrix

Back to top

Figure 51. Description of the task *Estimate suitability for each requirement*

The example of how the semi-product of the decision making might look is given in the Figure 52.



Example: Example of Pugh Matrix - suitability

Expand All Sections Collapse All Sections

Relationships

Related Elements	• Estimate suitability for each requirement

Back to top

Main Description

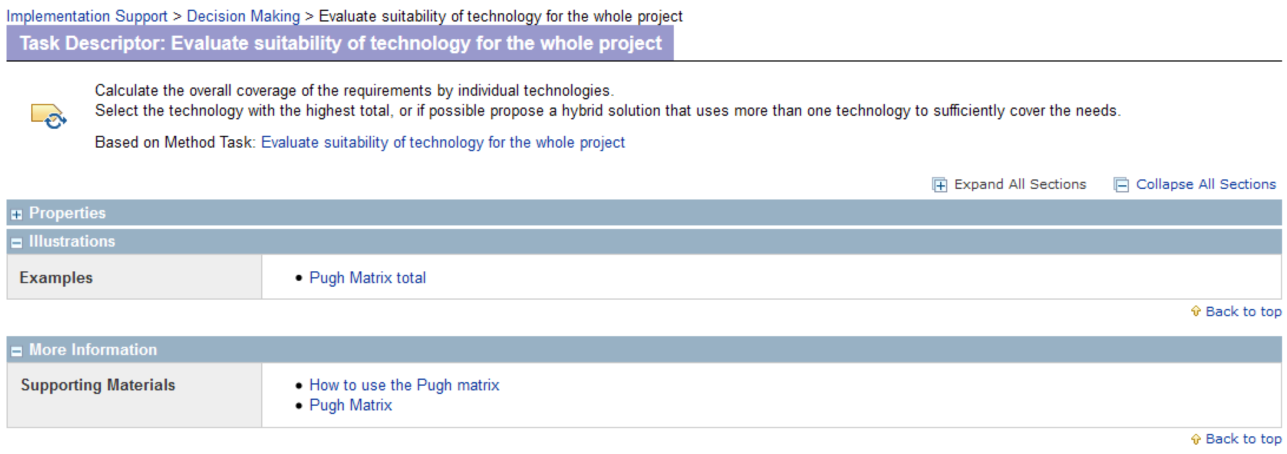
Example of Pugh matrix

Criterion	Weight	OSLC	Capra	Papyrus
Compatibility with legacy systems	1	*3*	*1*	2
Web-based	2	*2*	*3*	*3*
Mature/stable	3	*1*	*2*	*1*
TOTAL				

Back to top

Figure 52. Example of the Pugh matrix with estimated suitabilities

The decision making should be completed by the task described in the Figure 53.



Implementation Support > Decision Making > Evaluate suitability of technology for the whole project

Task Descriptor: Evaluate suitability of technology for the whole project

Calculate the overall coverage of the requirements by individual technologies.
Select the technology with the highest total, or if possible propose a hybrid solution that uses more than one technology to sufficiently cover the needs.

Based on Method Task: Evaluate suitability of technology for the whole project

Expand All Sections Collapse All Sections

Properties

Illustrations

Examples	• Pugh Matrix total

Back to top

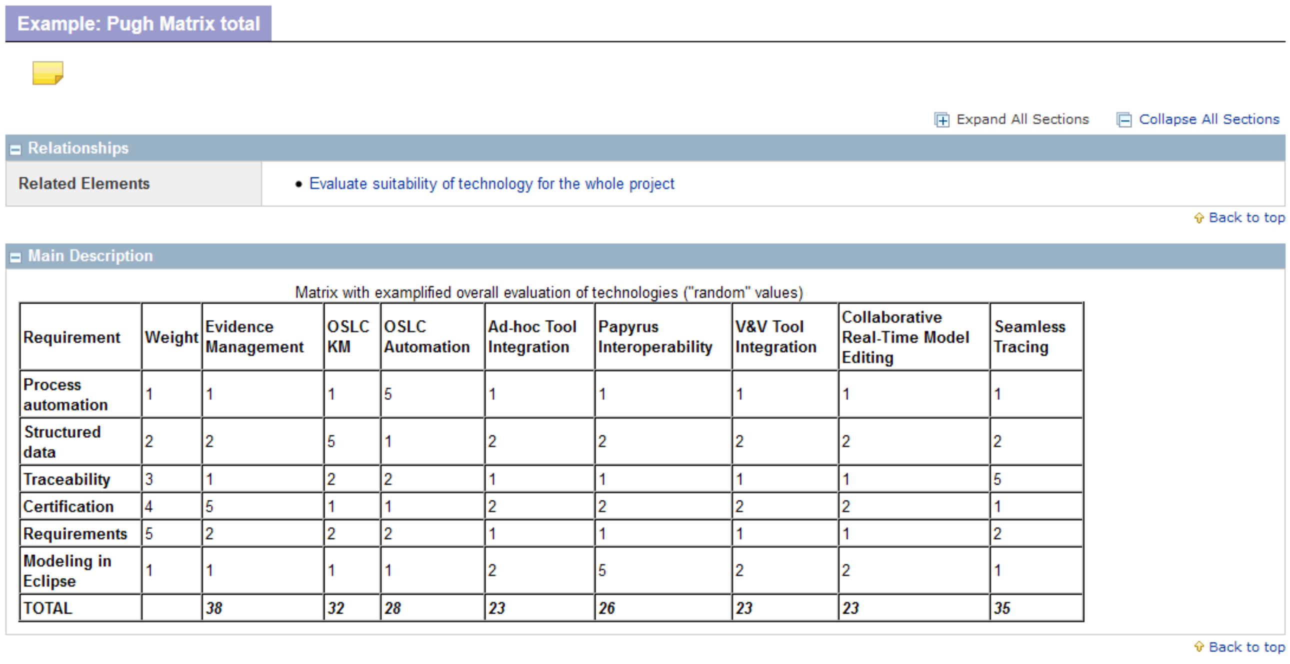
More Information

Supporting Materials	• How to use the Pugh matrix • Pugh Matrix

Back to top

Figure 53. Description of the task *Evaluate suitability of technology for the whole project*

An example of the resulting matrix with the overall evaluation of the suitability of individual approaches in the given context is provided by the Figure 54.



Example: Pugh Matrix total

Expand All Sections Collapse All Sections

Relationships

Related Elements	• Evaluate suitability of technology for the whole project

Back to top

Main Description

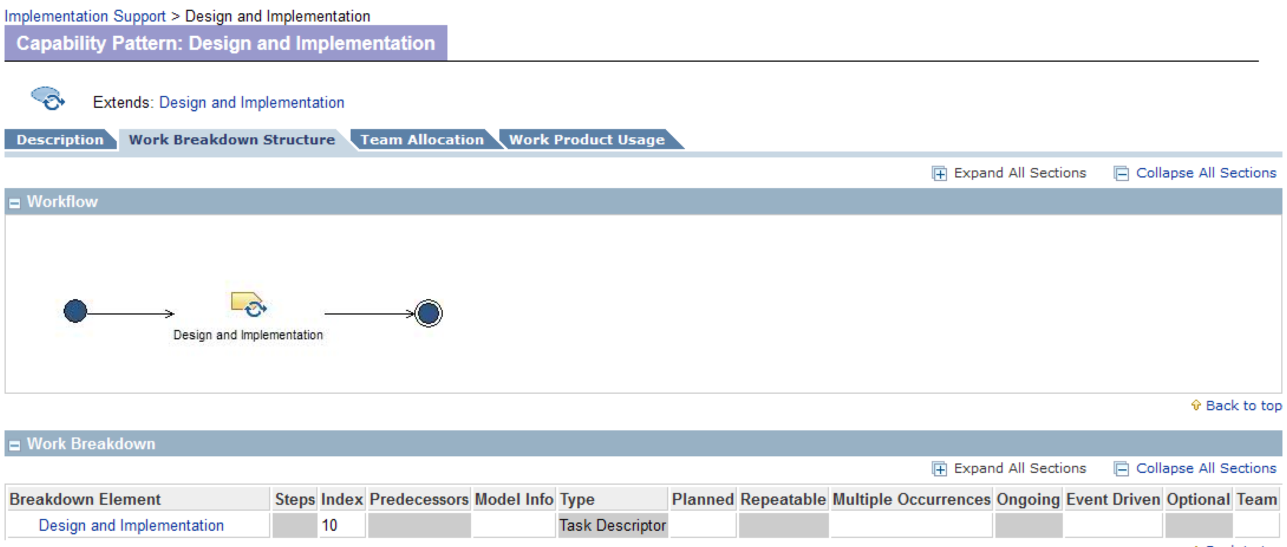
Matrix with examplified overall evaluation of technologies ("random" values)

Requirement	Weight	Evidence Management	OSLC KM	OSLC Automation	Ad-hoc Tool Integration	Papyrus Interoperability	V&V Tool Integration	Collaborative Real-Time Model Editing	Seamless Tracing
Process automation	1	1	1	5	1	1	1	1	1
Structured data	2	2	5	1	2	2	2	2	2
Traceability	3	1	2	2	1	1	1	1	5
Certification	4	5	1	1	2	2	2	2	1
Requirements	5	2	2	2	1	1	1	1	2
Modeling in Eclipse	1	1	1	1	2	5	2	2	1
TOTAL		*38*	*32*	*28*	*23*	*26*	*23*	*23*	*35*

Back to top

Figure 54. Example of the computed total technology values

The design and implementation activities referenced in the Figure 55 constitute a major effort. They are not described here in detail, since they are highly dependent on the chosen approach and on the context (the needs, the available tools). The choice of the approach is exemplified by the Figure 54.


Implementation Support > Design and Implementation

Capability Pattern: Design and Implementation

Extends: Design and Implementation

Description | Work Breakdown Structure | Team Allocation | Work Product Usage

Expand All Sections Collapse All Sections

Workflow

Back to top

Work Breakdown

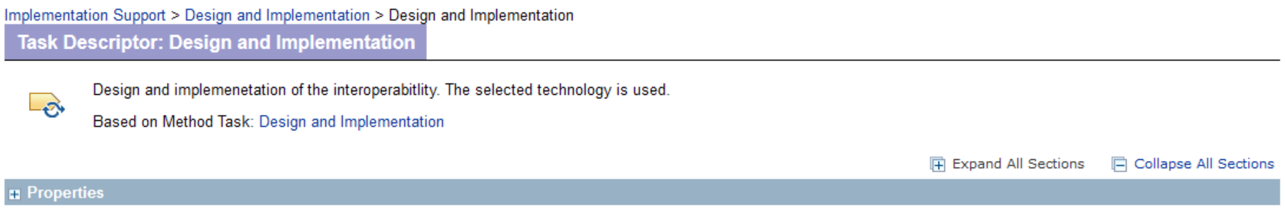
Expand All Sections Collapse All Sections

Breakdown Element	Steps	Index	Predecessors	Model Info	Type	Planned	Repeatable	Multiple Occurrences	Ongoing	Event Driven	Optional	Team
Design and Implementation		10			Task Descriptor							

Back to top

Figure 55. Activity diagram of *Design and Implementation*

The Figure 56 describes the intent of the activity depicted on the Figure 55. More detailed hints about the integration of tools can be found in the Section 3 of this deliverable.


Implementation Support > Design and Implementation > Design and Implementation

Task Descriptor: Design and Implementation

Design and implemenetation of the interoperabitlity. The selected technology is used.

Based on Method Task: Design and Implementation

Expand All Sections Collapse All Sections

Properties

Figure 56. Description of the task *Design and implementation*

Appendix B. The OSLC KM Resource Shape

B.1 Base Knowledge Management

Compliance

Table 9. Base Knowledge Management Compliance

Requirement	Level	Meaning
Unknown properties and content	MAY / MUST	OSLC services MAY ignore unknown content and OSLC clients MUST preserve unknown content
Resource Operations	MUST	OSLC service MUST support resource operations via standard HTTP operations
Resource Paging	MAY	OSLC services MAY provide paging for resources but only when specifically requested by service consumer
Partial Resource Representations	MUST / MAY	OSLC services MUST support request for a subset of a resource's properties via the oslc.properties URL parameter retrieval via HTTP GET and MAY support via HTTP PUT
Partial Update	MAY	OSLC services MAY support partial update of resources using patch semantics
Service Provider Resources	MAY / MUST	OSLC service providers MAY provide a Service Provider Catalog and MUST provide a Service Provider resource
Creation Factories	MUST / MAY	OSLC service providers MUST provide at least one creation factory resource for concepts, relationships, metaproperties, semantics and artefacts and MAY provide creation factory resources for collections of the aforementioned resources
Query Capabilities	MUST	OSLC service providers MUST provide query capabilities to enable clients to query for resources
Query Syntax	MUST	OSLC query capabilities MUST support the OSLC Core Query Syntax
Delegated UI Dialogs	MUST	OSLC Services MUST offer delegated UI dialogs (for both creation and selection) specified via service provider resource
UI Preview	SHOULD	OSLC Services SHOULD offer UI previews for resources that may be referenced by other resources
HTTP Basic Authentication	MAY	OSLC Services MAY support Basic Authentication and SHOULD only do so only over HTTPS
OAuth Authentication	MAY	OSLC Services MAY support OAuth and MAY indicate the required OAuth URLs via the service provider resource
Error Responses	MAY	OSLC Services MAY provide error responses using Core defined error formats
RDF/XML Representations	MUST	OSLC services MUST support RDF/XML representations for OSLC Defined Resources
XML Representations	MUST	OSLC services MUST support XML representations that conform to the OSLC Core Guidelines for XML
JSON Representations	MAY / MUST	OSLC services MAY support JSON representations; those which do MUST conform to the OSLC Core Guidelines for JSON
HTML Representations	MAY	OSLC services MAY provide HTML representations for GET requests

B.2 Specification Versioning

See Core Specification Version 2.0 - Specification Versioning.

Service providers that support the resource formats and services in this specification **MUST** add an HTTP response header of OSLC-Core-Version with a value of 2.0. Consumers **SHOULD** request formats and services defined in this document by providing a HTTP request header of OSLC-Core-Version with a value of 2.0. See section below on Version Compatibility with OSLC KM 1.0 Specifications.

This specification reserves, for possible future use, the use of the HTTP header OSLC-KM-Version. OSLC Providers **MUST NOT** use this HTTP header.

Namespaces

In addition to the namespace URIs and namespace prefixes oslc, rdf, dcterms and foaf defined in the Core Specification Version 2.0, OSLC KM defines the namespace URI of http://trc-research.github.io/spec/km/ with a preferred namespace prefix of oslc_km.

Furthermore, the SKOS (Simple Knowledge Organization System), a W3C Recommendation, is also defined through the namespace: http://www.w3.org/2004/02/skos/core and prefix: skos. Other semantic-based vocabularies will use the *de facto* namespace and prefix that can be searched using the service: Prefix.cc.

Resource Formats

In addition to the requirements for Core Specification Version 2.0 - OSLC Defined Resource Representations, this section outlines further refinements and restrictions.

For HTTP GET/PUT/POST requests on all OSLC KM and OSLC Core defined resource types,

- KM Providers **MUST** support RDF/XML representations with media-type application/rdf+xml. KM Consumers **MUST** be prepared to deal with any valid RDF/XML document.
- KM Providers **MUST** support XML representations with media-type application/xml. The XML representations **MUST** follow the guidelines outlined in Core Specification Appendix B: Representations and Examples.
- KM Providers **MAY** support JSON representations with media-type application/json. The JSON representations **MUST** follow the guidelines outlined in Core Specification Appendix B: Representations and Examples.

Additionally, for HTTP GET,

- KM Providers **SHOULD** provide an [X]HTML representation and a user interface (UI) preview as defined by Core Specification Version 2.0 UI Preview

For HTTP GET response formats for Query requests,

- KM Providers **MUST** support RDF/XML representations with meda-type application/rdf+xml.
- KM Providers **MUST** support XML representations with media-type application/xml.
- KM Providers **MAY** support JSON representations with media-type application/json.

OSLC Providers **MAY** refuse to accept RDF/XML documents which do not have a top-level rdf:RDF document element. The OSLC Core describes an example, non-normative algorithm for generating RDF/XML representations of OSLC Defined Resources.

In addition to the resource formats defined above, providers **MAY** support additional resource formats; the meaning and usage of these resource formats is not defined by this specification.

Authentication

See Core Specification Version 2.0 - Authentication. OSLC KM places no additional constraints on authentication.

Error Responses

See Core Specification Version 2.0 - Error Responses. OSLC KM places no additional constraints on error responses.

Pagination

OSLC KM service providers **SHOULD** support pagination of query results as defined by the OSLC Core Specification. OSLC KM service providers **MAY** support pagination of a single resource's properties as defined by the OSLC Core Specification.

Requesting Selected Properties

A client may want to request a subset of a resource's properties as well as properties from a referenced resource. In order to support this behaviour a service provider **MUST** support the oslc.properties and oslc.prefix URL parameter on a HTTP GET request on individual resource request or a collection of resources by query. If the oslc.properties parameter is omitted on the request, or if the value of this parameter is "*", then all resource properties **MUST** be provided in the response. See OSLC Core Specification - Selective Property Values.

Updating Selected Properties

A provide **MAY** accept oslc.properties on a PUT with the meaning that only that subset of the resource's properties be updated.

If the parameter oslc.properties contains a valid resource property on the request that is not provided in the content, the server **MUST** treat that as a request to remove that property from the resource. If the parameter oslc.properties contains an invalid resource property, then a 409 Conflict **MUST** be returned.

B.3 KM Resource Definitions

Property value types that are not defined in the following sections, are defined in Core Specification Version 2.0 - Defining OSLC Properties.

The meaning of the columns in the following Table 10 is defined as follows. See also OSLC Core Specification Appendix B: Common Properties for further details on Resource Shapes.

- Occurs: The multiplicity of the property (corresponds to "oslc:occurs" on an "oslc:Property" resource).
- Read-only: Whether the Provider will accept value changes (corresponds to "oslc:readOnly" on an "oslc:Property" resource). "Unspecified" indicates that this specification places no requirements on a Provider's behaviour in this regard.
- Value-type: Corresponds to "oslc:valueType" on an "oslc:Property" resource.
- Representation: Corresponds to "oslc:representation" on an "oslc:Property" resource.
- Range: Corresponds to "oslc:range" on an "oslc:Property" resource. "Any" indicates that this specification places no "oslc:range" constrains on a property. Consumers in particular should not make assumptions about the range of such properties.
- Description: A textual description of the meaning of the property.

KM Resources

Table 10. OSLC KM: System Representation Language resources

Class	Resource Shape Name	Description
Artefact	`oslc_km:Artefact`	A container of relationships between concepts and metaproperties to semantically describe any piece of information. It is the basis for the creation of an underlying semantic network.
Relationship	`oslc_km:Relationship`	A relationship represents a link between any set of resources. It is possible to add semantics and it can contain any number of elements representing binary, ternary or even n-ary relationships.
Data	`oslc_km:Data`	An attribute-value expression that represents a property of the artefact under description.
MetaData	`oslc_km:MetaData`	A tag-value attribute representing typical metadata properties. Dublin Core is used here to represent such information. Both can be any type of resource or, more specifically, concepts.
Term	`oslc_km:Concept`	This concept follows the semantics and shape of a *skos:Concept*. More specifically: "*the notion of a SKOS concept is useful when describing the conceptual or intellectual structure of a knowledge organization system, and when referring to specific ideas or meanings established within a KOS (Knowledge Organization System)*".
Type	`oslc_km:Concept`	Everything has a type and a type is a kind of concept coming from a classification. E.g. The types of UML metamodel, such as Class, Use Case, etc.

Artefact Resource

Table 11. OSLC KM: The Artefact resource shape

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
dcterms:identifier	Exactly-one	True	String	Inline	rdfs:Literal	The unique identifier for this artefact.
dcterms:title	One-or-many	True	String	Inline	rdfs:Literal	The title of the artefact used to display a name.
dcterms:description	Zero-or-many	False	String	Inline	rdfs:Literal	The long description of this artefact that must be explanatory enough to understand what the artefact contains and is used to.
dcterms:created	Exactly-one	True	DateTime	Inline	xsd:dateTimeStamp	The date and time in which the artefact was created. The range is restricted to a data time stamp, although the Dublin Core allows us to use any rdfs:Literal. See: http://dublincore.org/documents/dcmi-terms/#terms-created
dcterms:modified	Zero-or-many	False	DateTime	Inline	xsd:dateTimeStamp	The moment in which the artefact was modified or redefined. The range is restricted to a data time stamp, although the Dublin Core allows us to use any rdfs:Literal. See: http://dublincore.org/documents/dcmi-terms/#terms-created
dcterms:creator	One-or-many	True	Resource	Reference	foaf:Agent	The agents (people, organizations or tools) that have defined this artefact.
oslc_km:term	Zero-or-one	False	Either Resource or Local Resource	Either Reference or Inline	oslc_km:Concept	The lexical form of this artefact (apart from title and description). It is an URI to a concept.
oslc_km:artefact-type	Zero-or-one	True	Either Resource or Local Resource	Either Reference or Inline	oslc_km:Concept	A link to a concept describing the type of this artefact. E.g. "Class Diagram"
oslc_km:relationships	Exactly-one	False	Either Resource or Local Resource	Either Reference or Inline	rdf:List	A list of relationships between the concepts within the artefact. Similar to skos:member (actually it is a kind of syntax sugar and the meaning of this property and skos:member is the same).
oslc_km:metaproperties	Zero-or-one	False	Either Resource or Local Resource	Either Reference or Inline	Rdf:List	A list of metaproperties for this artefact identifed by tag and value. It is a kind of wrapper for two concepts.

oslc_km:owned-artefacts	Zero-or-one	False	Either Resource or Local Resource	Either Reference or Inline	rdf:List	A list of artefacts that belongs to this artefact. It is similar to skos:member and skos:inScheme but with artefacts instead of concept schemes.
oslc_km:alt-visualization	Zero-or-many	False	Resource	Reference	N/A (Not applicable)	The alternative visual representation of this artefact using SVG+CSS.
oslc_km:preferred-visualization	Zero-or-one	False	Resource	Reference	N/A	The preferred visual representation of this artefact using SVG+CSS.
oslc_km:interpretation	Zero-or-one	False	Resource	Either Reference or Inline	N/A	A complete interpretation of this artefact through a concept description. E.g. Class diagram, etc.
oslc_km:traced-by	Zero-or-many	False	Resource	Either Reference or Inline	Rdf:Resource	A resource that traces this artefact.
oslc_km:traces-to	Zero-or-many	False	Resource	Either Reference or Inline	Rdf:Resource	A resource that is being traced by this artefact.
oslc_km:trace-type	Exactly-one	True	Resource	Either Reference or Inline	oslc_km:Concept	A link to a concept that explains how the trace has been created, etc. This element must be linked to the "trace" node (if any).
oslc_kpi:dataset	Zero-or-many	True	Resource	Reference	Qb:Dataset	The link to the datasets that contain observations that can affect this artefact. E.g. if a requirement is an artefact, the requirements quality observations would be the dataset linked to the artefact in a certain moment of time.
dcterms:source	Zero-or-many	True	Resource	Reference	Rdf:Resource	The set of documents that explains why this artefact should be explained.
oslc_km:access	Zero-or-one	False	Either Resource or Local Resource	Either Reference or Inline	N/A	A link to a resource describing how to access to a HTTP-based resource for gathering contents and convert into an artefact. The W3C HTTP vocabulary (a W3C note) is used to represent the information of an HTTP request.
oslc:valueShape	Exactly-one	False	Resource	Either Reference or Inline	rdf:Resource	A link to an URI that contains the shape of this artefact.
oslc_km:contents	Zero-or-one	False	String	Inline	rdfs:Literal	A literal representing the contents of any artefact in RDF. These contents are interpreted following the shape that must be also presented in the description of the artefact.

oslc_km:sparql-endpoint	Zero-or-one	False	Resource	Either Reference or Inline	xsd:anyURI	An URI pointing to a SPARQL endpoint from which the contents of an artefact will be gathered through a DESCRIBE query.

Metadata/Data Resource

Table 12. OSLC KM: The Metaproperty resource shape

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
dcterms:identifier	Exactly-one	True	String	Inline	rdfs:Literal	The unique identifier for this metaproperty
oslc_km:tag	Exactly-one	False	Either Resource or Local Resource	Either Reference or Inline	oslc_km:Concept	A tag for this metaproperty represented through a concept or even any resource.
oslc_km:value	Zero-or-one	False	Either Resource or Local Resource	Either Reference or Inline	oslc_km:Concept	A value for this metaproperty represented through a concept or even any resource.

Relationship Resource

Table 13. OSLC KM: The Relationship resource shape

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
dcterms:identifier	Exactly-one	True	String	Inline	rdfs:Literal	The unique identifier for this relationship. It is now an string but it would be better a skos:Concept to avoid broken links between pieces of data.
oslc_km:semantics	Zero-or-one	True	Either Resource or Local Resource	Either Reference or Inline	rdf:Property	The concept (property) that represents the semantics of this relationship.
oslc_km:from	Zero-or-one	False	Resource	Either Reference or Inline	rdf:List	The list of concepts from which a relationship is created. It is similar to skos:member but a new name used to provide a more meaningful name. Status: the name of this property is still open.
oslc_km:to	Zero-or-one	False	Resource	Either Reference or Inline	rdf:List	The list of concepts to which a relationship is created. It is similar to skos:member but a new name used to provide a more meaningful name. Status: the name of this property is still open.

Concept Resource

Table 14. OSLC KM: The Concept resource shape

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
skos:altLabel	Zero-or-many	False	String	Inline	rdf:PlainLiteral	"The preferred and alternative labels are useful when generating or creating human-readable representations of a knowledge organization system. These labels provide the strongest clues as to the meaning of a SKOS concept." Source: http://www.w3.org/TR/skos-reference/#labels
skos:broadMatch	Zero-or-many	False	Resource	Reference	skos:Concept	This property is used to state mapping (alignment) links between SKOS concepts in different concept schemes, where the links are inherent in the meaning of the linked concepts. Source: http://www.w3.org/TR/skos-reference/#L4307
skos:broader	Zero-or-many	False	Resource	Reference	skos:Concept	It is a semantic relation used to assert a direct hierarchical link between two SKOS concepts. Source: http://www.w3.org/TR/skos-reference/#broader
skos:broaderTransitive	Zero-or-many	False	Resource	Reference	skos:Concept	It is a property used to assert a direct hierarchical link between two SKOS concepts. More specifically, it is used to both direct and indirect hierarchical links between concepts. It is the transitive version of skos:broader. Source: http://www.w3.org/TR/skos-reference/#broaderTransitive
skos:changeNote	Zero-or-many	False	String	Inline	rdf:PlainLiteral	It is an annotation property. According to the SKOS recommendation: "There is no restriction on the nature of this information, e.g., it could be plain text, hypertext, or an image; it could be a definition, information about the scope of a concept, editorial information, or any other type of information. ". Source: http://www.w3.org/TR/skos-reference/#notes
skos:closeMatch	Zero-or-many	False	Resource	Reference	skos:Concept	It is used to link two concepts that are sufficiently similar that they can be used interchangeably in some information retrieval applications. In order to avoid the possibility of "compound errors" when combining mappings across more than two concept schemes, skos:closeMatch is not

Prefixed Name	Occurs	Read -only	Value-type	Represent ation	Range	Description
						declared to be a transitive property. Source: http://www.w3.org/TR/skos-reference/#L4307
skos:definition	Zero-or-many	False	String	Inline	rdf:PlainLiteral	It is an annotation property. According to the SKOS recommendation: "There is no restriction on the nature of this information, e.g., it could be plain text, hypertext, or an image; it could be a definition, information about the scope of a concept, editorial information, or any other type of information. ". Source: http://www.w3.org/TR/skos-reference/#notes
skos:editorialNote	Zero-or-many	False	String	Inline	rdf:PlainLiteral	It is an annotation property. According to the SKOS recommendation: "There is no restriction on the nature of this information, e.g., it could be plain text, hypertext, or an image; it could be a definition, information about the scope of a concept, editorial information, or any other type of information. ". Source: http://www.w3.org/TR/skos-reference/#notes
skos:exactMatch	Zero-or-many	False	Resource	Reference	skos:Concept	This property is used to link two concepts, indicating a high degree of confidence that the concepts can be used interchangeably across a wide range of information retrieval applications. It is a transitive property, and is a sub-property of close match. Source: http://www.w3.org/TR/skos-reference/#L4307
skos:example	Zero-or-many	False	String	Inline	rdf:PlainLiteral	It is an annotation property. According to the SKOS recommendation: "There is no restriction on the nature of this information, e.g., it could be plain text, hypertext, or an image; it could be a definition, information about the scope of a concept, editorial information, or any other type of information. ". Source: http://www.w3.org/TR/skos-reference/#notes
skos:hiddenLabel	Zero-or-many	False	String	Inline	rdf:PlainLiteral	It is a property to label concepts. Source: http://www.w3.org/TR/skos-reference/#labels
skos:historyNote	Zero-or-many	False	String	Inline	rdf:PlainLiteral	It is an annotation property. According to the SKOS recommendation: "There is no restriction on the nature of this

Prefixed Name	Occurs	Read -only	Value-type	Represent ation	Range	Description
						information, e.g., it could be plain text, hypertext, or an image; it could be a definition, information about the scope of a concept, editorial information, or any other type of information. ". Source: http://www.w3.org/TR/skos-reference/#notes
skos:inScheme	Zero-or-many	False	Resource	Reference	skos:ConceptScheme	The scheme (an aggregation of one or more SKOS concepts) to which the concept belongs. Source: http://www.w3.org/TR/skos-reference/#schemes
skos:mappingRelation	Zero-or-many	False	Resource	Reference	skos:Concept	It is a mapping property to link concepts. It is the superclass of other mapping properties. Source: http://www.w3.org/TR/skos-reference/#mapping
skos:narrowMatch	Zero-or-many	False	Resource	Reference	skos:Concept	This property is used to state mapping (alignment) links between SKOS concepts in different concept schemes, where the links are inherent in the meaning of the linked concepts. Source: http://www.w3.org/TR/skos-reference/#mapping
skos:narrower	Zero-or-many	False	Resource	Reference	skos:Concept	It is a semantic relation used to assert a direct hierarchical link between two SKOS concepts. Source: http://www.w3.org/TR/skos-reference/#broader
skos:narrowerTransitive	Zero-or-many	False	Resource	Reference	skos:Concept	It is a property used to assert a direct hierarchical link between two SKOS concepts. More specifically, it is used to both direct and indirect hierarchical links between concepts. It is the transitive version of skos:broader. Source: http://www.w3.org/TR/skos-reference/#broaderTransitive
skos:notation	Zero-or-many	False	Resource	Either Reference or Inline	rdf:PlainLiteral	It is an annotation property. According to the SKOS recommendation: "There is no restriction on the nature of this information, e.g., it could be plain text, hypertext, or an image; it could be a definition, information about the scope of a concept, editorial information, or any other type of information. ". Source: http://www.w3.org/TR/skos-reference/#notes
skos:prefLabel	Zero-or-	False	String	Inline	rdf:Plain	The preferred and alternative labels are

Prefixed Name	Occurs	Read-only	Value-type	Representation	Range	Description
	many				Literal	useful when generating or creating human-readable representations of a knowledge organization system. These labels provide the strongest clues as to the meaning of a SKOS concept. "A resource has no more than one value of skos:prefLabel per language tag." Source: http://www.w3.org/TR/skos-reference/#labels
skos:related	Zero-or-one	False	Resource	Reference	skos:Concept	The property skos:related is used to assert an associative link between two SKOS concepts. Source: http://www.w3.org/TR/skos-reference/#semantic-relations
skos:relatedMatch	Zero-or-many	False	String	Reference	skos:Concept	This property is used to state mapping (alignment) links between SKOS concepts in different concept schemes, where the links are inherent in the meaning of the linked concepts. More specifically, it is used to state an associative mapping link between two concepts.
skos:scopeNote	Zero-or-many	False	String	Inline	rdf:PlainLiteral	It is an annotation property. According to the SKOS recommendation: "There is no restriction on the nature of this information, e.g., it could be plain text, hypertext, or an image; it could be a definition, information about the scope of a concept, editorial information, or any other type of information. ". Source: http://www.w3.org/TR/skos-reference/#notes
skos:semanticRelation	Zero-or-many	False	String	Reference	skos:Concept	It is the super property of all mapping and relationship properties. It is used to assert generic semantic relationships between concepts.
skos:topConceptOf	Zero-or-many	True	Resource	Reference	skos:ConceptScheme	It serves to state that a concept is a root of a concept scheme. Source: http://www.w3.org/TR/skos-reference/#schemes
dcterms:creator	One-or-many	True	Resource	Either Reference or Inline	foaf:Agent	The agents (people, organizations or tools) that have defined this concept.
dcterms:contributor	Zero-or-many	False	Resource	Either Reference or Inline	foaf:Agent	The agents (people, organizations or tools) that have contributed to the definition of this concept.
dcterms:created	Exactly-	True	DateTime	Inline	xsd:dateTimeSta	The time in which this concept has been

Prefixed Name	Occurs	Read -only	Value-type	Represent ation	Range	Description
	one				mp	created.
dcterms:modified	Zero-or-many	False	DateTime	Inline	xsd:dateTimeStamp	The moment in which this concept has been modified or redefined.
skos:memberList	Exactly-one	False	Either Resource or Local Resource	Either Reference or Inline	rdf:List	A list of skos concepts (ordered collection) that serves to specify the components of the pattern.
dcterms:identifier	Exactly-one	True	String	Inline	xsd:string	The unique identifier for this concept.

Relationship Labels

When a KM relationship property is to be presented in a user interface, it may be helpful to provide an informative and useful textual label for that relationship instance. (This in addition to the relationship property URI and the object resource URI, which are also candidates for presentation to a user.) To this end, OSLC providers **MAY** suppport a dcterms:title link property in RM resource representations where a relationship property is permitted, using the anchor approach outlined in the OSLC Core Links Guidance.

Providers and consumers should be aware that the dcterms:title of a link is unrelated to the dcterms:title of the object resource. Indeed, links may carry other properties with names in common to the object of the link, but there is no specified relationship between these property values.

B.4 KM Service Provider Capabilities

Service Provider Resources

Service providers **MUST** provide one or more oslc:ServiceProvider resources as defined by Core Specification Version 2.0 - Service Provider Resource. Discovery of OSLC Service Provider Resources **MAY** be via one or more OSLC Service Provider Catalog Resources, or may be discovered by some other and/or additional Provider-specific means outwith the scope of this specification. The oslc:Service resources referenced by this oslc:ServiceProvider **MUST** have an oslc:domain of http://trc-research.github.io/spec/km/.

Service providers **MAY** provide one more more oslc:ServiceProviderCatalog resources as defined by Core Specification Version 2.0 - Service Provider Resources. Any such catalog resources **MUST** include at least one oslc:domain of http://trc-research.github.io/spec/km/. Discovery of top-level OSLC Service Provider Catalog Resources is outwith the scope of this specification.

Service providers **MUST** give an oslc:serviceProvider property on all OSLC Defined Resources. This property **MUST** refer to an appropriate oslc:ServiceProvider resource.

Creation Factories

Service providers supporting resource creation **MUST** do so through oslc:CreationFactory resources, as defined by Core Specification Version 2.0 - Creation Factories. Any such factory resources **MUST** be discoverable through oslc:Service resources. Providers **SHOULD** provide oslc:ResourceShape resources on oslc:CreationFactory resources as defined by OSLC Core Specification Appendix B: Common Properties - Resource Shapes.

Query Capabilities

Service providers **MUST** support query capabilities, as defined by Core Specification Version 2.0 - Query Capabilities. Providers **SHOULD** provide oslc:ResourceShape on oslc:QueryCapability resources as defined by OSLC Core Specification Appendix B: Common Properties - Resource Shapes.

The Query Capability **MUST** support these parameters:

- oslc.where
- oslc.select
- oslc.properties
- oslc.prefix

Where oslc:ResourceShape is not supported by the Query Capability, providers **SHOULD** use the following guidance to represent query results:

- For RDF/XML and XML, use rdf:Description and rdfs:member as defined by Core Specification Appendix B:Representations and Examples - RDF/XML Examples.
- For JSON the query results are contained within oslc:results array. See Core Specification Appendix B: Representations and Examples - Guidelines for JSON.

The stability of query results is OPTIONAL (see Core Specification Version 2.0 - Stable Paging).

Delegated UIs

OSLC KM service providers **MUST** support the selection and creation of resources by delegated web-based user interface dialogs Delegated UIs as defined by OSLC Core.

OSLC KM service providers **MAY** support the pre-filling of creation dialogs based on the definition at Delegated UIs.

Usage Identifiers

OSLC KM service provider **MAY** identify the usage of various services with additional property values for the OSLC Core defined oslc:usage property on oslc:Dialog, CreationFactory and QueryCapability. The oslc:usage property value of http://open-services.net/ns/core#default **SHOULD** be used to designate the default or primary service to be used by consumers when multiple entries are found.

There are no additional usage identifiers defined by this specification. OSLC Providers **MAY** provide their own usage URIs. Such usage URIs **MUST** be in a non-OSLC namespace.

Media Types

To identify a format of RDF/XML, the media type used for KM resource representations **MUST** be application/rdf+xml. The usage of the OSLC KM 1.0 defined media types of application/x-oslc-km-artefact-1.0+xml, application/x-oslc-km-artefact-collection-1.0+xml, application/x-oslc-km-service-description-1.0+xml and application/x-oslc-disc-service-provider-catalog+xml is deprecated.

Requesting formats

KM 1.0 consumers wanting to request 1.0 resource formats will not need to change if they used 1.0 defined media types (application/x-oslc-km*). KM 1.0 consumers should use media types as defined in this specification for requests, excluding the OSLC KM 1.0 specific media types (application/x-oslc-km*). KM consumers supporting should request request 1.0 media types on HTTP GET requests as usually done with

HTTP request parameter Accept giving appropriate quality (See HTTP Accept) weighting to help distinguish their preferred content.

For additional guidance, a KM 1.0 consumer or provider **MAY** reference the OSLC-Core-Version HTTP header with a value of 2.0.

B.5 Open Issues

As it has been outlined in previous sections, the main objective of this specification is to provide a way for representing any piece of knowledge using the SRL model. Since there are a lot of techniques for knowledge representation, it is important to emphasize that the use of SRL model is motivated because:

1. It has been specially designed for information retrieval purposes and
2. it is fully supported in the Knowledge Manager tool.

However, and with the aim of keeping backward compatibility, a mapping to existing RDF data has been also presented and implemented. This approach allows us to provide a mechanism for those that want to publish RDF data for which there is no shape or vocabulary (SRL could be used) and to enable a way of re-using existing RDF data sources. Nevertheless, the transformation of RDF to SRL has been designed at a graph level so a higher type of transformation (keeping logic formalisms if any) is under study. For instance, an RDFS (RDF Schema) and OWL (Ontology Web Language), W3C standards for ontology construction, mapping to SRL are ongoing work.

On the other hand, this specification can be also seen as a broader effort, containing certain parts of existing specifications such as Asset Management and Tracked Resource Set. In this case, these specifications should be merged reusing the existing concepts and properties. Furthermore, and in order to support a full knowledge management strategy, the OSLC KM could be extended to:

- Support a kind of formal reasoning or underlying logic formalism.
- Include more provenance information. E.g. W3C Provenance Ontology.
- Expose more services such as traceability of quality checking of any artefact.
- Expose a general-purpose visualization service.

Appendix C : Document changes with respect to D5.7

New Sections:

Section	*Title*
2.8	Collaborative Editing
2.9	Safety/Cyber Architect Tools Integration
2.10	Data and Security Management
3.2.6	Use in the AMASS Platform
3.8	Collaborative Editing
3.9	Safety/Cyber Architect Tools Integration
3.10	Data and Security Management
Appendix C	Document changes with respect to D5.7

Modified Sections:

Section	*Title*	*Change*
	Executive Summary	Minor update
2.2	OSLC KM	Minor update
2.2.1	Mapping between any piece of RDF to the OSLC KM Data Shape	Minor update
2.3	V&V Manager and OSLC Automation	Minor update
3.2.5	Limitations and Lessons Learnt	Minor update
3.3.1	SysML elements and corresponding OSLC properties	Minor update
3.3.2	Public Verification Server	Minor update
3.5	Papyrus Interoperability	Added description about where to find the Papyrus features.
3.6	V&V Tool Integration	Added description of the integration of xSAP tool
4	Conclusions	Minor update
B.3	KM Resource Definitions	Minor update