ECSEL Research and Innovation actions (RIA)

AMASS

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

Prototype for Cross-Domain and Intra-Domain Reuse (c)

D6.6

<table>
<thead>
<tr>
<th>Work Package:</th>
<th>WP6 Cross-Domain and Intra-Domain Reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dissemination level:</td>
<td>PU = Public</td>
</tr>
<tr>
<td>Status:</td>
<td>Final</td>
</tr>
<tr>
<td>Date:</td>
<td>31 October 2018</td>
</tr>
<tr>
<td>Responsible partner:</td>
<td>Borja López (TRC)</td>
</tr>
<tr>
<td>Contact information:</td>
<td><a href="mailto:borja.lopez@reusecompany.com">borja.lopez@reusecompany.com</a></td>
</tr>
<tr>
<td>Document reference:</td>
<td>AMASS_D6.6_WP6_TRC_V1.0</td>
</tr>
</tbody>
</table>

PROPRIETARY RIGHTS STATEMENT
This document contains information that is proprietary to the AMASS consortium. Permission to reproduce any content for non-commercial purposes is granted, provided that this document and the AMASS project are credited as source.
### Contributors

<table>
<thead>
<tr>
<th>Names</th>
<th>Organisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Borja López, Luis M. Alonso</td>
<td>The REUSE Company (TRC)</td>
</tr>
<tr>
<td>Ángel López, Huáscar Espinoza, Alejandra Ruiz</td>
<td>Tecnalia Research &amp; Innovation (TEC)</td>
</tr>
<tr>
<td>Barbara Gallina, Inmaculada Ayala, Faiz Ul Muram, Muhammad Atif Javed, Irfan Sljivo, Julieth Castellanos</td>
<td>Maelardalens Hoegskola (MDH)</td>
</tr>
<tr>
<td>Stefano Puri</td>
<td>Intecs (INT)</td>
</tr>
<tr>
<td>Jose María Álvarez, Roy Mendieta, Miguel Rozalen, Fabio di Ninno</td>
<td>Universidad Carlos III de Madrid (UC3)</td>
</tr>
</tbody>
</table>

### Reviewers

<table>
<thead>
<tr>
<th>Names</th>
<th>Organisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helmut Martin (Peer-reviewer)</td>
<td>Virtual Vehicle (VIF)</td>
</tr>
<tr>
<td>Morayo Adedjouma (Peer-reviewer)</td>
<td>Commissariat a L’energie Atomique et aux Energies Alternatives (CEA)</td>
</tr>
<tr>
<td>Cristina Martinez (Quality Manager)</td>
<td>Tecnalia Research &amp; Innovation (TEC)</td>
</tr>
<tr>
<td>Garazi Juez (TC review)</td>
<td>Tecnalia Research &amp; Innovation (TEC)</td>
</tr>
<tr>
<td>Stefano Puri (TC review)</td>
<td>Intecs (INT)</td>
</tr>
</tbody>
</table>
## TABLE OF CONTENTS

**Executive Summary** .............................................................................................................................. 7

1. **Introduction** ........................................................................................................................................ 8

2. **Implemented Functionality** .................................................................................................................. 10
   2.1 Scope .................................................................................................................................................. 10
   2.2 Implemented Requirements .................................................................................................................. 11
      2.2.1 Modelling of standards .................................................................................................................. 13
      2.2.2 Tailoring of Standards models to specific projects ........................................................................ 18
      2.2.3 Process Compliance (informal) management .................................................................................. 20
      2.2.4 Solution for Process Compliance Checking .................................................................................... 26
      2.2.5 Compliance Monitoring .............................................................................................................. 31
      2.2.6 Reuse Assistance .......................................................................................................................... 31
      2.2.7 Product/Process/Assurance Case Line Specification .................................................................... 39
      2.2.8 Reuse Discovery ........................................................................................................................... 49
   2.3 Installation and User Manuals ............................................................................................................... 56

3. **Implementation Description** ............................................................................................................... 57
   3.1 Implemented Modules ......................................................................................................................... 57
      3.1.1 Compliance Management ............................................................................................................ 57
      3.1.2 Management of families/lines ...................................................................................................... 57
      3.1.3 Cross/Intra Domain Reuse .......................................................................................................... 59
   3.2 Source Code Description for the AMASS Tool Platform........................................................................ 59
   3.3 Source Code Description for External Tools ....................................................................................... 65
      3.3.1 OSLC-KM integration .................................................................................................................... 65
      3.3.2 Ontology Reuse Operations ........................................................................................................ 65

4. **Conclusions** .......................................................................................................................................... 66

**Abbreviations and Definitions** ................................................................................................................. 67

**References** ............................................................................................................................................. 69

**Appendix A: Document changes with respect to D6.5** ............................................................................. 71
Figure 1. AMASS Building blocks in AMASS Prototype P2 .........................................................8
Figure 2. Functional decomposition for the AMASS platform ....................................................10
Figure 3. Standard and Process Modelling ................................................................................14
Figure 4. Reference Framework Editor ......................................................................................14
Figure 5. ISO 26262: Recommendation Table associated to a Requirement ..........................15
Figure 6. Authoring perspective of the EPF composer ...............................................................16
Figure 7. Standard modelled in the EPF composer .....................................................................17
Figure 8. Recommendation table modelled in the EPF composer ..............................................17
Figure 9. Tailoring of Baseline Models from Standard Models ................................................18
Figure 10. Baseline tailoring .......................................................................................................19
Figure 11. Baseline graphical editor ..........................................................................................19
Figure 12. Assurance Project and System Component Specification structure ..........................21
Figure 13. How to create Compliance Maps ...............................................................................22
Figure 14. Compliance Set view ................................................................................................22
Figure 15. Mapping Table view ..................................................................................................23
Figure 16. Import View in OpenCert for EPF: Result of the import operation .........................24
Figure 17. Modelling of compliance mappings in the EPF composer ....................................25
Figure 18. Mapped Requirements in the EPF composer .............................................................25
Figure 19. Generating baseline model from EPF mapping requirements ..................................26
Figure 20. Plugins required for compliance checking ...............................................................26
Figure 21. Custom Categories ....................................................................................................27
Figure 22. Rule specification .......................................................................................................27
Figure 23. Rule set specification ...............................................................................................27
Figure 24. Process Elements Plugin ..........................................................................................28
Figure 25. Annotated process description plugin .......................................................................28
Figure 26. Annotating a task with compliance effects ...............................................................28
Figure 27. The work breakdown structure of the delivery process ............................................29
Figure 28. Activity diagram of the delivery process .................................................................29
Figure 29. Compliance checking result provided by Regorous ..................................................30
Figure 30. Compliance report in web client ................................................................................31
Figure 31. Cross-project Reuse View ..........................................................................................32
Figure 32. Using the Reuse View ..............................................................................................33
Figure 33. Context menu to choose a search technology ............................................................34
Figure 34. Window to introduce the search parameters .............................................................34
Figure 35. Results of a search (right one with automatic selection option active) ....................35
Figure 36. How to create Equivalence Map. ..............................................................................36
Figure 37. Equivalence Map form ............................................................................................37
Figure 38. Cross-Domain button ..............................................................................................37
Figure 39. Cross-domain window ..............................................................................................38
Figure 40. The achievement of error free models ....................................................................40
Figure 41. ECSS-E-ST-40C compliant SoPL ...........................................................................42
Figure 42. Detecting fallacies in process models menu .............................................................45
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Requirements implemented in the first prototype of the AMASS platform (Core Prototype)</td>
<td>11</td>
</tr>
<tr>
<td>Table 2</td>
<td>Requirements implemented in the second prototype of the AMASS platform (Prototype P1)</td>
<td>11</td>
</tr>
<tr>
<td>Table 3</td>
<td>Requirements implemented in the third prototype of the AMASS platform (Prototype P2)</td>
<td>13</td>
</tr>
</tbody>
</table>
Executive Summary

The deliverable D6.6 – Implementation for Cross Domain and Intra Domain Reuse (c), is the third output of the task T6.3 (Implementation for Cross-Domain and Intra-Domain Reuse). Based on the results of tasks T2.2 (AMASS Reference Tool Architecture and Integration) and T6.2 (Conceptual Approach for Cross-Domain and Intra-Domain Reuse), task T6.3 develops a prototype-tooling framework to support cross and intra-domain reuse, as well as compliance management. D6.6 is the evolution of D6.5, which described the second prototype.

This deliverable reports the final status of the tooling framework for the AMASS platform by describing the supported WP6-related functionality and the details about its implementation.

T6.3 progresses iteratively and incrementally, in close connection with the conceptual task (T6.2) and the other technical WPs (WP2 to WP5). The implementation follows the requirements of the case studies, which must benchmark the prototypes. This iterative process helps benchmarking the prototype implementation, so that it makes easier to check the continuous refinement done during the different AMASS prototypes.

The WP6-related part of the final iteration (P2) of the AMASS platform extends the initial implementation of the basic building blocks for this prototype, which has been a consolidation and integration of results from previous projects. More concretely, the developed tools in the WP6-related part of P2 support the following functional areas:

- Capture, retrieve and share information from standards.
- Define compliance and equivalence mappings.
- Generate argumentation fragments based on development processes.
- Manage assurance projects.
- Monitor progress status of assurance project.
- Reuse discovery and reuse assistance.
- Variability management at assurance case/process/product level.
- Semi-automatic generation of product/process arguments.

This document presents in detail the pieces of functionality implemented in the AMASS platform for the areas above, their software architecture, the technology used, and source code references.

D6.6 relates to other AMASS outcomes referred in this deliverable:

- D6.5, where the main differences are described in the chapter Appendix A: Document changes with respect to D6.5.
- Installable AMASS platform tools for the Prototype P2.
- User manual and installation instructions [18].
- Source code [17].
1. Introduction

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 approaches proposed by the EU projects OPENCOSS [1] and SafeCer [2] 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 (Open Services for Lifecycle Collaboration) specifications [12]).

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 industrialisation. The Open Community will be supported by a governance board, and by rules, policies, and quality models. This includes support for the AMASS base tools (tool infrastructure for database and access management, among others) and extension tools enriching the AMASS platform functionalities. As Eclipse Foundation is part of the AMASS consortium, the Polarsys/Eclipse community ([www.polarsys.org](http://www.polarsys.org)) is going to host the 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 Building blocks](image)

**Figure 1.** AMASS Building blocks in AMASS Prototype P2
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), were aligned, merged and consolidated at Technology Readiness Levels TRL4\(^1\). Concerning this first prototype, the basic building block assigned to WP6 was Compliance Management.

2. During the **second prototyping** iteration (Prototype P1), the AMASS-specific Building Blocks were developed and benchmarked at TRL4; this comprises the blue basic building blocks as well as the green building blocks in Figure 1. Regarding WP6, in this second prototype the specific building blocks included a (cross/intra-domain) reuse assistant potentially using semantics standards equivalence mappings and a toolset for product/process/assurance case line specification.

3. Finally, at the **third prototyping** iteration (Prototype P2), all AMASS building blocks are integrated in a comprehensive toolset operating at TRL5. WP6 functionalities developed during the second prototype iteration are enhanced and fully integrated with functionalities of 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 using the tool-supported solutions. The case studies are described in D1.1 [20].

As part of the Prototype P2, WP6 is responsible for driving the work resulting on intra-domain and cross-domain reuse, in order to design and implement the building blocks for “Reuse Assistant”, “Impact Analysis”, “Automatic Generation of Process/Product based Arguments”, “Semantics Standards Equivalence Mapping” and “Product/Process/Assurance Case reuse via management of variability” (cf. Figure 1).

This deliverable reports the **tool development** results of the building blocks commented above. It presents in detail the design of the functionality implemented in the AMASS platform tools, the software architecture, the technology used, and source code references. The design is based on the investigated state-of-the-art and state-of-practice approaches. Their gaps are identified to come up with a way forward, enabling the formulation of requirements to achieve the reuse-oriented vision of AMASS. This activity will serve to ensure both, the innovation of the project and future feasibility of exploitation of results.

Other important documents related to D6.6 are:

- Installable AMASS Platform tools for the Prototype P2.
- User manuals and Installation instructions [18].
- Source code.

---

\(^1\) 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](http://ec.europa.eu/research/participants/data/ref/h2020/other/wp/2016_2017/annexes/h2020-wp1617-annex-g-trl_en.pdf)
2. Implemented Functionality

2.1 Scope

As stated in Section 1, the building blocks assigned to WP6 in this prototype are “Reuse Assistant”, “Semantics Standards Equivalence Mapping” and “Product/Process/Assurance Case Line Specification”. Also, some improvements have been implemented regarding the “Compliance Management” and “Assurance Project Lifecycle Management” blocks (see Figure 2).

These blocks are highlighted with a red circle in Figure 2, showing the general functional overview of the AMASS platform (from deliverable D3.6 [14]).

Furthermore, WP6 is aimed at providing the framework for cross-domain and intra-domain reuse.

The Compliance Management building block has been improved in this third prototype to allow monitoring the compliance progress using filtering by criticality levels (e.g., Safety Integrity Level, Automotive Safety Integrity Level, Development Assurance Level). The compliance is also performed by the generation of a compliance report in a web client application.

The AMASS platform supports the Compliance Management functionalities with two toolsets:

- Tools from the OpenCert project².
- Tools from the EPF (Eclipse Process Framework) project³.

---

² Further information about the OPENCOSS toolset can be found at www.opencoss-project.eu and https://www.polarsys.org/projects/polarsys.opencert

³ Further information about the EPF toolset can be found at http://www.eclipse.org/epf/
The following section details both the satisfied requirements and the deployed components to show the implementation scope of the WP6-related part of the Prototype P2.

## 2.2 Implemented Requirements

From the requirements point of view, D6.6 has focused on a set of AMASS requirements as defined in deliverable D2.1 [13]. The requirements listed in Table 1 were implemented in the first iteration of the WP6-related part of the AMASS platform (Prototype Core). The column "Related requirement" refers to the list of items gathered in the deliverable D2.1.

**Table 1. Requirements implemented in the first prototype of the AMASS platform (Core Prototype)**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Related Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling of standards</td>
<td>WP6_CM_001</td>
<td>The AMASS tools shall be able to model a set of industrial standards (including the parts, activities, requirements, work products, and criticality levels from the standards).</td>
</tr>
<tr>
<td>Tailoring of standards models to specific projects</td>
<td>WP6_CM_002</td>
<td>The AMASS tools shall enable the tailoring of standards models to a specific project (e.g., by establishing the parts of the standard that apply to a given assurance project).</td>
</tr>
<tr>
<td>Compliance Monitoring</td>
<td>WP6_CM_005</td>
<td>The AMASS tools shall support web-based monitoring of compliance status to be filtered by any custom criteria.</td>
</tr>
<tr>
<td>Process Compliance (informal) management</td>
<td>WP6_CM_008</td>
<td>The AMASS tools shall enable users to visualise process compliance. This means showing the links between the requirements and the applicant’s evidence (during the planning as well as execution phase). This visualisation could be done via compliance maps (matrix) or via arguments aimed at justifying the satisfaction of the requirements coming from the standards.</td>
</tr>
</tbody>
</table>

In addition to that, the Prototype P1 focused on a second set of AMASS requirements as defined in deliverable D2.1 [13].

**Table 2. Requirements implemented in the second prototype of the AMASS platform (Prototype P1)**

<table>
<thead>
<tr>
<th>Function name</th>
<th>Related Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intra-Domain, Intra standard, Reuse Assistance</td>
<td>WP6_RA_001</td>
<td>The AMASS tools shall enable partial reuse of compliance artefacts when transiting from one project to another (different criticality level, etc.). The commonality that characterises the different projects should be recognised and proposed as reusable process structure.</td>
</tr>
<tr>
<td>Intra-Domain, Cross standards, Reuse Assistance</td>
<td>WP6_RA_002</td>
<td>The AMASS tools shall enable partial reuse of compliance artefacts when transiting from one project to another (different/same criticality level, if applicable, but different standards (e.g., Automotive SPICE, ISO 26262)). The commonality that characterises the different projects should be recognised and proposed as a reusable process structure.</td>
</tr>
<tr>
<td>Function name</td>
<td>Related Requirement</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Intra-Domain, Cross versions, Reuse Assistance</td>
<td>WP6_RA_003</td>
<td>The AMASS tools shall enable partial reuse of compliance artefacts when transiting from one project to another (different/same criticality level, if applicable, but different standards or different versions of a standard (e.g., ISO 26262-2011, ISO 26262-2018)). The commonality that characterises the different projects should be recognized and proposed as reusable process structure.</td>
</tr>
<tr>
<td>Cross-Domain Reuse Assistance</td>
<td>WP6_RA_004</td>
<td>The AMASS tools shall enable partial reuse of compliance artefacts when transiting from one project to another belonging to different domains (e.g., from automotive to avionics). The commonality that characterises the different projects should be recognised and proposed as reusable process structure.</td>
</tr>
<tr>
<td>Intra-Domain, Intra standard, Different Stakeholders, Reuse/Integration Assistance</td>
<td>WP6_RA_005</td>
<td>The AMASS tools shall enable partial reuse of compliance artefacts during the integration (manufacturer/supplier). Assumed process requirements vs. actual process requirements.</td>
</tr>
<tr>
<td>The AMASS tools must support variability management at process level</td>
<td>WP6_PPA_001</td>
<td>The AMASS tools shall enable the specification/systematisation of variability at the process level.</td>
</tr>
<tr>
<td>The AMASS tools must support specification of variability at the component level</td>
<td>WP6_PPA_004</td>
<td>The system shall enable users to specify what varies (and what remains unchanged) from one component and its evolved version at component level.</td>
</tr>
<tr>
<td>The AMASS tools must support variability management at the assurance case level</td>
<td>WP6_PPA_005</td>
<td>The system shall enable users to specify what varies (and what remains unchanged) from one component and its evolved version at component level.</td>
</tr>
<tr>
<td>Semi-automatic generation of product arguments</td>
<td>WP6_PPA_002</td>
<td>Assurance case arguments. This could be done by enabling semi-automatic generation of product-based arguments-fragments.</td>
</tr>
<tr>
<td>Semi-automatic generation of process arguments</td>
<td>WP6_PPA_003</td>
<td>The system shall reduce efforts of manual creation of process-based assurance case arguments. This could be done by enabling semi-automatic generation of process-based arguments-fragments.</td>
</tr>
<tr>
<td>Process Compliance (formal) management</td>
<td>WP6_CM_009</td>
<td>The AMASS tools shall enable users to formally check process compliance.</td>
</tr>
</tbody>
</table>
Finally, this last phase (Prototype P2) focuses on the last iteration of the AMASS requirements as defined in deliverable D2.1 [13]. The requirements listed in Table 3 have been implemented during the third iteration of the WP6-related part of the AMASS platform. The column “Related requirement” refers to the list of items gathered in the deliverable D2.1.

### Table 3. Requirements implemented in the third prototype of the AMASS platform (Prototype P2)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Related Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triggering compliance Checking</td>
<td>WP6_CM_004</td>
<td>The AMASS tools shall provide the functionality for automatically triggering the requirements for (re)checking the compliance of safety processes against rules – especially, when there are changes in the standards/regulations.</td>
</tr>
<tr>
<td>Process Compliance (formal) management)</td>
<td>WP6_CM_009</td>
<td>The AMASS tools shall enable users to formally check process compliance.</td>
</tr>
<tr>
<td>Reusable off the shelf components</td>
<td>WP6_RA_006</td>
<td>The AMASS tool shall provide the capability for reuse of pre-developed components and their accompanying artefacts.</td>
</tr>
<tr>
<td>Semantics-based mapping of standards</td>
<td>WP6_SEM_001</td>
<td>The AMASS tools shall enable the mapping of standards based on their semantics.</td>
</tr>
<tr>
<td>The AMASS tools must support variability management at the assurance case level</td>
<td>WP6_PPA_005</td>
<td>The system shall enable users to specify what varies (and what remains unchanged) from one component and its evolved version at component level.</td>
</tr>
</tbody>
</table>

Each requirement, together with the implementation done so far that implements the requirement, is shortly outlined in the following sections.

#### 2.2.1 Modelling of standards

This chapter describes the implementation of the requirement **WP6_CM_001**, described in Table 1.

This feature is supported by both the OpenCert [26] and EPF toolsets [27]. Figure 3 shows examples of models for these two toolsets.

- **OpenCert** uses Reference Framework Editor (as part of the OpenCert tools) to model standards (IEC 61508, ISO 26262, DO-178C, EN 50126, and the like) and regulations (either as additional requirements or model elements in a given model representing a standard or a new Reference Framework). Each Reference Framework model can be also mapped to others Reference Framework models by using the concept of Equivalence Map (diamond form).

- **EPF** can be used to model company-specific processes (e.g., the process at Alstom or Thales to develop safety-critical systems).
3.1.2.1 Reference Framework Editor

The Reference Framework Editor is composed of five views (see Figure 4):

1. The **Repository Explorer** view shows the contents of the repository.
2. The **Outline** view shows the elements of the model and permits its edition.
3. The **Diagram Editor** permits the graphical modelling of a subset of concepts of the Reference Framework. The Diagram Editor can be replaced by the **Tree View Editor**, which is opened by double clicking on the file “.refframework” in the Repository Explorer.
4. The **Palette** view is a toolbox with the concepts of the model and the connections between them to add to the diagram.
5. The **Properties** view is used to edit the properties of the element of the selected model.

In addition, the recommendation or applicability tables (that is, the SIL levels) from industry standards can be specified using the Tree View editor, by associating the Tables to specific Requirements or Activities. Figure 5 illustrates an example for ISO 26262 recommendation table. A similar approach can be used for other standards such as DO-178C objective tables.
3.1.2.2 Standards modelling with EPF

The EPF composer is a tool for the modelling of engineering process based on the SPEM 2.0 (Software & Systems Process Engineering Metamodel) OMG (Object Management Group) standard [25]. The functionality of the EPF composer is organised in two views, the Authoring perspective (opened by default in the EPF Composer) and the Browsing perspective. The goal of the Authoring perspective is to provide functionality to formally model process element and processes, while the goal of the Browsing perspective is to present the contents modelled of the Authoring perspective. So, most of the work of the user will take place in this last perspective.

Figure 6 shows a screenshot of this modelling perspective, which is composed of three parts: the Method library (left top of the workbench), the Configuration (left bottom of the workbench) and the Process element/process modelling space (right part of the workbench) that, in this case, is showing the modelling of a delivery process.
Standards can be modelled in the EPF composer tool [23] following an approach similar to those described in [22] for the IBM Rational Method Composer. In the work presented in [22], standards are modelled using a new user defined type named Requirement. However, EPF does not support the definition of a new user defined type. In order to overcome this limitation, we have customized the guidance “Practice” with an icon and variability relationships making possible the application of the mentioned work. The EPF composer supports variability mechanisms that are at disposal in SPEM 2.0. These variability mechanisms focus on set semantic relationships between process elements of the same type. These semantics relationships make possible to define a new process element as a variation of an existing one.

Practices in EPF represent a proven way or strategy of doing work to achieve a goal that has a positive impact on work product or process quality. They are usually used to group process elements that belong to some practice like risk management, software quality verification or component-based development just to mention a few. Therefore, in our view, practice semantics and use are aligned with the semantics of requirements. Then, standards are modelled in the Authoring perspective of the EPF composer as a collection of nested Requirements (i.e. customised practices) (see Figure 7).
The EPF composer also supports the modelling of the recommendation tables by means of customized practices. In this case, five customized practices were created to represent tables, criticality levels, recommendation levels and concepts to make possible to associate these three concepts. Recommendation tables are modelled as a composition of customized practices of different kinds. Figure 8 shows the result of modelling of a recommendation table from RTCA DO-178C.

The EPF composer allows import and export projects (known as plug-ins in EPF) in the library space. This allows using process and process elements defined in the imported project in other projects of the library space. In order to facilitate the modelling of standard information in the EPF composer, we have grouped...
all the modelling concepts that we have developed for the modelling of standard information in a plug-in that can be imported in any method library.

### 2.2.2 Tailoring of Standards models to specific projects

This chapter describes the implementation for requirement **WP6_CM_002**, described in Table 1. The information managed by the AMASS tools can be organised in two types: project-independent information that can be used by various projects (e.g., models of generic processes and standards) and project-specific information (e.g., evidence and argumentation models). The main element of project-specific information is the so-called Assurance Project. One important part of an Assurance Project is the Baseline model. A Baseline model represents what is planned to comply with (regarding a Reference Framework model) a specific Assurance Project. Baseline models can be tailored from Reference Framework models.

As shown in Figure 9, Baseline models can be instantiated from the Reference Framework models. It is possible to import Baseline models based on the Standard processes modelled in EPF.

![Tailoring of Baseline Models from Standard Models](image)

**Figure 9.** Tailoring of Baseline Models from Standard Models

Each baseline model results from importing (copying) a Reference Framework model and selecting the subset of activities, artefacts and the like that apply in a given Assurance Project. Figure 10 shows the selection step before creating a baseline model upon a reference framework model (the latter displayed in a tree view).
Figure 10. Baseline tailoring

We have implemented a filter so that the elements affected by the criticality and the applicability levels of the standard are selected accordingly. Figure 11 shows an example of baseline model automatically generated from the reference framework model.

Figure 11. Baseline graphical editor

Transformation of Standard Requirements to Baseline Model: Model-Driven Engineering principles and techniques can facilitate and improve assurance of safety-critical systems by providing support for the transformation of standard requirements modelled (through the customization of the guidance “Practice”) in EPF Composer to Baseline models in OpenCert. The mapping between Unified Method Architecture
(UMA) metamodel [27] and Baseline metamodel is achieved by using Epsilon Transformation Language (ETL) [8]. For that, a plugin has implemented that transform the standard requirements to Baseline models. The main mapping rules for baseline generation are as follows:

- Content Package of EPF Composer is mapped into Base Framework
- Requirements (modelled as practices) are mapped into Base Requirements
- Id, name and description of requirements are mapped into Id, name and description

2.2.3 Process Compliance (informal) management

This chapter describes the implementation for requirement WP6_CM_008, described in Table 1. As described in Section 2.2.2, users can maintain the lifecycle of projects by creating Assurance Projects. Figure 12 illustrates the elements of an Assurance Project. An Assurance Project has three main elements:

1. Baseline Configuration. A Baseline Configuration has a set of Baseline Models. A Baseline model represents what is prescribed in a specific assurance project.
2. Permissions Configuration. This functionality has been revised for Prototype P2. It supports profile creation to enable restricted access to AMASS functionality and data.
3. Assurance Assets Package. This is a pointer to project-specific Artefacts models, Argumentation models, Process models and System models. These four models represent what has been done in a specific assurance project. System Component models are managed by the CHESS toolset (see deliverable D3.4 [15]). The implementation plan for this point has been revised for P2 due to CDO technical problems during P1. Dependencies with external projects have prevented us to make the link for this prototype. We expect to achieve the problems in the last prototype.

The mapping of Assurance Asset Package elements with Baseline Models is specified using the concept of Compliance Map.

Additionally, Evidence and Process models can be created using EPF process planning models (model transformation arrows in Figure 12). This model transformation helps users get a first version of their evidence and execute process models to demonstrate compliance with standards.
3.1.2.3 **Compliance modelling with OpenCert**

Compliance maps can be edited in two different views that can be opened from the Baseline models, as shown in Figure 13. “Mapping Set” allows users to edit each of the compliance maps by using a tree view. “Mapping Table” shows a summary of the compliance maps and their status in the form of a table.
Figure 13. How to create Compliance Maps

Figure 14 shows the Compliance Set view.

The Compliance Set view form is organised in three zones:

- The **left zone** shows the baseline model elements.
- The **middle zone** allows to make different filters and to add the type of mapping (Full, Partial, No Map) and any mapping justification.
- The **right zone** shows the list of target models and their model elements (evidence, argumentation or process).
It is possible to create compliance maps for activities, artefacts, requirements, roles and techniques, with the following allowed maps:

- BaseArtefact -> Artefact
- BaseRequirement -> Artefact , Claim or Activity
- BaseActivity -> Activity
- BaseRole -> Participant
- BaseTechnique -> Technique

Figure 15 shows the Compliance Table view. The Compliance Map window is organised in two zones:

- The upper part (blue circle) has controls to allow filtering. It is possible to filter by criticality level, applicability level, map model, a map group of the selected map model, a type of element that could be mapped and the mapping type. The “Not Defined” option is used to include in the table all the elements of the baseline that have not mapping established; in this way, it is possible to see the compliance Gap. It is necessary to click the Search button to begin the search process based in the filter options selected that will fill the table.

- The lower part (green circle) shows three controls:
  - The compliance mapping table that shows all the baseline elements that accomplish with the searching criteria selected by the user. By default, all the baseline elements of a compliance map are shown in the table.
  - A text box that shows the compliance map justification introduced for the base element selected in the bottom left table with a simple left click.
  - A list that shows all the target elements of the base element selected in the table with a simple left click.

![Figure 15. Mapping Table view](image-url)
**Importing EPF process models into OpenCert for compliance management**

It is also possible to export the EPF planned process models and import them in OpenCert. As abovementioned, EPF can be used for modelling the definition and planning of processes. OpenCert also allows users to model processes but looking at post-planning phases. We can get benefit of the EPF process information to create a first view (which can evolve during an assurance project) of process models in OpenCert by importing EPF information into OpenCert. Specifically, the transformation process takes a delivery process modelled in EPF and generates an evidence model and a process model in OpenCert.

Figure 16 shows the Importing View implemented in OpenCert.

![Import View in OpenCert for EPF](https://via.placeholder.com/150)

**Figure 16. Import View in OpenCert for EPF: Result of the import operation**

### 3.1.2.4 Compliance mappings in EPF

The compliance of a process with a specific standard can be modelled in the EPF composer as proposed in [22]. The use of EPF for compliance modelling allows planning how our process will address standard requirements.

As a part of the approach presented in [22], it is necessary to define a new method plug-in in the EPF composer that will contain just the requirements of the standard mapped to elements of a process. This procedure makes possible to re-use standards and processes for different compliance mappings. Compliance mappings are modelled in the references tab of the requirements (see Figure 17). In this tab, we can use as evidence for compliance process activities, a portion of a process (i.e. a capability of a pattern) and guidance elements like guidelines, tools or practices. Activities include actions, roles and work products involved in the activity. EPF includes filters and patterns to make the selection of evidence for compliance easier.
This modelling solution supports three types of compliance: full compliance, partial compliance and no-compliance. The full compliance is modelled providing at least one evidence for requirement like the case of “Document Architecture Alternative” (see Figure 18). The partial compliance is modelled by decomposing requirements in sub-requirements and providing evidence just for the part of the requirements that is accomplished. This is illustrated by the requirement “Document sys requirements”: the sub-requirement “Detail a use case” is fulfilled by the process with the activity “Detail use case scenarios”, while the sub-requirement “Identify and outline requirements” is not fulfilled. Therefore, “Document sys requirements” is partially satisfied by the process. Finally, “Development Environment Multi Part” does not have any associated evidence, so the process does not address this requirement.

The transformation of standard’s requirements modelled in EPF Composer (compliant with UMA metamodel) to baseline models (compliant with CACM metamodel) is supported (see Figure 19). The generated models will be stored in selected assurance case project in the CDO repository.
2.2.4 Solution for Process Compliance Checking

This chapter describes the implementation for the requirements WP6_CM_009 described in the Table 3. The compliance checking vision is supported by EPF composer [27] and Regorous [33]. To be able to check compliance, Regorous requires three models, namely the execution semantics of the process, the compliance effects annotations, and the rule set. To provide the process descriptions required by Regorous, the creation of three plugins, as done by the IBM approach for mapping standards requirements [34], is the methodology adopted. The three plugins can be seen in Figure 20, and are explained below.

1. **Plugin for capturing standard’s requirements**: In the method authoring of EPF composer, we capture the standard's requirements using custom categories. The root of the custom categories is the name of the standard. The novelty added to this plugin is that the rule set and the rules are added by using a
customized reusable asset for the former and customized concepts for the latter. The rules are associated to the corresponding standard’s requirements.

![Custom Categories](image)

**Figure 21. Custom Categories**

The rule is written in the main description field of the compliance effect, as presented in Figure 22.

![Rule specification](image)

**Figure 22. Rule specification**

The rule set is defined in a customized reusable asset, which contains the superiority relations between rules, as presented in Figure 23.

![Rule set specification](image)

**Figure 23. Rule set specification**

2. **Plugin for capturing process elements**: In the method authoring of EPF composer, we capture the process elements required to support the software process modeling by using the current modeling capabilities of EPF composer (see Figure 24).
3. **Plugin for annotating process description**: This plugin is used to match standard’s requirements with processes. This plugin contains an extended copy of the tasks defined in the previous plugin (by using contributes to the original ones) in the method authoring of EPF composer (see the tasks that are surrounded by a red box in Figure 25).

   ![Process Elements Plugin](image)
   
   **Figure 24.** Process Elements Plugin

   Then, tasks are annotated with compliance effects. To annotate the task, we double click on the task to open the task descriptor. Then we click on the tab Guidance and add the compliance effect annotations as presented in Figure 26.

   **Task: design_software_unit (Contributes to 'Design Software Unit' in 'Process Elements')**

   ![Guidance](image)
   
   **Figure 26.** Annotating a task with compliance effects
Once the tasks are annotated, we model the delivery process, by inserting the annotated tasks in the work breakdown structure, as presented in Figure 27.

<table>
<thead>
<tr>
<th>Presentation Name</th>
<th>Index</th>
<th>Predecessors</th>
<th>Model Info</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Software Unit Design</td>
<td>1</td>
<td></td>
<td></td>
<td>Task Description</td>
</tr>
<tr>
<td>Specify Software Unit Design</td>
<td>2</td>
<td>1</td>
<td></td>
<td>Task Description</td>
</tr>
<tr>
<td>Design Software Unit</td>
<td>3</td>
<td>2</td>
<td></td>
<td>Task Description</td>
</tr>
</tbody>
</table>

**Figure 27.** The work breakdown structure of the delivery process

Then, an activity diagram is created by using the proprietary activity diagram provided by EPF composer (see Figure 28).

![Activity diagram of the delivery process](image)

**Figure 28.** Activity diagram of the delivery process

Once the plugins are modelled, we export them by using the function “export XML file” provided by EPF composer. From the exported plugins, two files are extracted:

- First, an XMI file (usually called diagram), which describes the activity diagram, is selected for transferring the process description required by Regorous. The elements of interest are an Activity that provides the name of the process, an initial node and a final node that represent the start and end event respectively, one Activity parameter node for every task and one control flow for every sequence. Other process elements of interest could be a decision, merge, and fork and join nodes for modeling exclusive and parallel gateway respectively, which can be useful for complex processes.

- Second, an XML file, which provides the compliance annotated process information, is also extracted. In the activity name corresponds to the process name. Tasks have associated concepts that correspond to the compliance effects. We can also create the rule set since every concept is described with the actual rule and the reusable asset with the superiority relation.

These two files are transformed to the three models required by Regorous. First, the rule set can be obtained from the Delivery Process created in the plugin for the annotated process description (provided by EPF Composer). The corresponding mapping descriptions are listed below.

- Reusable Asset, a type of content element, is transformed into the rule set. The attributes transferred are name, presentationName and briefDescription.
- Concept, a type of content element, is transformed into the Terms. The attribute transferred is name.
- Each content category that contains a rule in the field brief description is transformed into a rule. The attributes transferred are name, presentationName, and briefDescription.

Regorous also requires the representation of the process in a canonical form (CPF). Within AMASS, such representation is given via EPF Composer-supported representation, which is based on UML 2.0 Diagram Interchange Specification. Below, the mapping between CPF and UML is given.

- Activity information is transformed to a canonical process in CPF. The attribute transferred is id.
- The Initial Node becomes a node with type start event in CPF.
- Each Activity Parameter Node becomes a task type in the CPF. Attributes transferred are id and name.
- Each Control Flow becomes an edge in CPF. Attributes transferred are id, name, source and target.
• The Activity Final Node becomes an end event type in CPF.
• The Decision Node becomes an XORSplitType in CPF.
• The Merge Node becomes an XORJoinType in CPF.
• The Fork Node becomes an ANDSplitType in CPF.
• The Join Node becomes an ANDJoinType in CPF.

Finally, the compliance effects annotations require a structure that complies with the Regorous schema called Compliance Check Annotations. This information can be retrieved from EPF Composer taking into account that the process elements can be extracted from the process structure (described with UML elements) and the compliance effects annotations can be extracted from the delivery process (described with UMA elements). The corresponding matching elements description is listed below.

• A reusable asset becomes a ruleSetList. The attribute transferred is the name.
• Each edge becomes a special element in the compliance annotations file called conditions. The attribute transferred is the id.
• The node becomes a Task Effects. The attribute transferred is the id. Then, the id is also used to search for the concepts that should be converted into the compliance effects in the delivery process file.
• Every concept associated to the task is transferred to the Effect. The attribute transferred is the name.

The generated files are provided as input to Regorous, which is able to check compliance and generate a compliance report as is shown in Figure 29.

**Compliance Check Results**

⚠️ Process is non-compliant.

### Process Warnings

The warnings below indicate structural issues with the process or potential issues with annotations that may affect the correctness of compliance check results.

<table>
<thead>
<tr>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information only (3 items)</td>
<td></td>
</tr>
<tr>
<td>Rule '2.1' was not invoked</td>
<td></td>
</tr>
<tr>
<td>Unknown term 'performProvideSwArchitecture'</td>
<td></td>
</tr>
<tr>
<td>Rule '3.2' was not invoked</td>
<td></td>
</tr>
</tbody>
</table>

### Non-compliant Execution Paths

Non-compliant execution paths and the cause of non-compliance are listed below.

- Unfulfilled obligation to 'Not produceSoftwareComponentDesign' (Achievement, non-
- Unfulfilled obligation to 'Not produceSoftwareComponentDesign' (Achievement, non-
- Unfulfilled obligation to 'Not produceSoftwareComponentDesign' (Achievement, non-
- Unfulfilled obligation to 'Not produceSoftwareComponentDesign' (Achievement, non-

**Figure 29.** Compliance checking result provided by Regorous
2.2.5 Compliance Monitoring

This chapter describes the implementation for requirement **WP6_CM_005**, described in Table 1.

Compliance report provides extensive functionality that helps the AMASS platform users to assess the current compliance of their project to the selected safety standard (i.e., baseline). Two modes of the report can be distinguished:

- An **interactive mode**, where the user can actively browse the report, select the specific baseline items, view their properties, their compliance mapping and the associated evidence, and add or remove the evidence resources mapped to the specific baseline element.
- A **printer friendly report** - which is a textual output presenting all the information of the current compliance of the selected project.

Figure 30 shows an example of compliance report in interactive mode. The “**Project Compliance**” table, which is placed in the left, presents base artefacts and base activities of the selected standard. The most important column is the “**Compliance Status**” one, which presents the overall compliance status of a project to the specific standard item. This column can be sorted by value, thus allowing user to assess the project compliance at one glance. In case base activities or base artefacts are defined to have a parent-child hierarchy, this relation is presented accordingly in a tree structure of the table.

In addition, users can look at the list of Argumentation, Evidence and Process model elements in the respective menu options at the top right corner of the web view.

![Figure 30. Compliance report in web client](image)

2.2.6 Reuse Assistance

This chapter describes the implementation for requirements **WP6_RA_001, WP6_RA_002, WP6_RA_003, WP6_RA_004, and WP6_RA_005**, described in the Table 2.

The reuse assistance functionality concerns intra and cross-domain reuse of assurance and certification assets. The Reuse Assistant functionality includes cross-system reuse and cross-standard reuse, as described in the following subsections.
### 3.1.2.5 Cross-systems reuse (intra standard or intra domain product upgrade)

This functionality implies reuse of assurance assets when a product or system evolves in terms of functionality or technology, e.g. product upgrade. Product upgrade corresponds to a development scenario in which an already-assessed system is modified and thus a new assessment (e.g., re-certification) is required. For example, a new system can be developed based on an existing one. Such a new system can include, for instance, some new components. We assume that the reusable assurance assets were compliant with the same standards we target in the new scenario.

We have implemented this functionality by using a specific view called “Reuse” in OpenCert. It is used to reuse models from one source assurance project to a target assurance project. This view is particularly useful to reuse a subset of model elements, which can be selected manually by the user.

Figure 31 and Figure 32 show how to open the Reuse View and how to select model elements to be reused. The Reuse view is also integrated in the set of OpenCert views and it will be already opened when the OpenCert perspective is activated.

![Cross-project Reuse View](image)

**Figure 31.** Cross-project Reuse View
The AMASS Prototype P2 version allows users through a context menu (see Figure 33) search a specific subset of an evidence model elements using two reuse discovery approaches. The first one is related to OSLC-KM module (see chapter Reuse Discovery based on OSLC-KM) and the second one related to Elastic Search (see Reuse Discovery based on ElasticSearch).

The user can search reusable assets in the selected evidence model according a text and/or a selected critically level and/or a selected applicability (see Figure 34). Those criticality and applicability levels are specified by the standard that the source project is compliant with. If the user activates the bottom option, the elements to reuse will be select automatically according the results of the search, modifying the previous subset of model elements selected.
Figure 33. Context menu to choose a search technology

Figure 34. Window to introduce the search parameters

The results are shown to the user with a colour code (see Figure 35).
• In green. The previously selected asset, by the user or by a previous search, is a good candidate for reusing according to the introduced searching criteria.

• In red. The previously selected asset, by the user asset or by a previous search, does not comply with the introduced searching criteria and therefore should be unselected to avoid its reuse.

• In yellow. The previously not selected asset, by the user asset or by a previous search, is a good candidate for reusing and therefore should be selected to be reused.

Figure 35. Results of a search (right one with automatic selection option active)

3.1.2.6 Cross-standard reuse (cross-concern or cross-domain)

This functionality is related to the reuse of assurance assets from a project that was completed in compliance with a different dependability concern (e.g., security-compliant assurance project reused from a safety-compliant assurance project) or different domain (e.g. avionics-compliant assurance project reused from an automotive compliant assurance project). The second standard could correspond to a new standard, a new version of a standard, or a different interpretation of a standard (e.g., by a different certification authority).

For cross-standard reuse, AMASS enables reuse of assurance assets of one assurance project in another project, when they relate to different industry domains, dependability concerns or industry standards in general. To perform cross-standard reuse, an Equivalence Map model must be created between the source and the target standard models. The Reuse Assistant provides information on the reuse opportunities as result of the equivalence relationships. Once the actor selects the assurance assets to be reused, the reuse operation itself can be executed. A module for compliance gap analysis allows AMASS users to look at the reuse post-conditions identified in the equivalence map model.

To create Equivalence Maps, we have developed a tailored functionality in OpenCert. To open it, we provide a menu called “Mapping Set” on the properties form of the reference framework using the tree view editor (see Figure 36).
**Figure 36. How to create Equivalence Map.**

Figure 37 shows the form for Equivalence Map. The Equivalence Map form is organised in three zones:

- **The left zone** shows the actual reference framework, and it loads the type of elements for which we want to make the equivalence maps.

- **The middle zone** allows to make different filters like:
  - *Filter Mapping Model* lists all the mapping models stored in the database. It will be necessary to select one of them and one group model.
  - *Filter Map Element*. It is possible to create equivalence maps for activities, artefacts, requirements, roles and techniques.
  - *Filter Equivalence Map*. This filter allows making different equivalence maps for the same refframework element.
  - The user must also introduce the mapping information in the middle part; this information consists of the ID, the name, the type and a justification text.

- **The right zone** shows two lists and a combo box.
  - *The combo box* shows all the database refframeworks to select the reference framework that will be the target of the equivalence map to create.
  - The *upper list* loads the elements, according to the filter selected, of the refframework chosen in the combo box that will be the target of the equivalence map to create.
  - The *lower list* displays the full content (not filtered) of the source refframework that will be postConditions in case of reusing. The postConditions are mandatory extra activities, not included in the standard, that must be performed in case of reusing the target element from one assurance project based in the target refframework in another assurance project based in the source refframework using the cross-domain functionality.
For the reuse assistant, it is mandatory that the target assurance project is based on a refframework with equivalence maps associated to the refframework, in which the source assurance project is based. Figure 38 shows how to open the reuse assistant for cross-domain reuse. We focus on the evidence reuse for this version of AMASS prototype.

Figure 37. Equivalence Map form.

Figure 38. Cross-Domain button
Figure 39 shows the cross-domain assistant form. It is organised in three zones:

- The left zone shows information about the target project. In the top part, the URL of the target assurance project can be found. Below, a tree shows the contents of the target baseline, where we can select one of the items in order to display its compliance map information in the tree at the bottom.

- The middle zone displays equivalence map information. It includes controls to select the equivalence mapping model and the equivalence map group and displays the equivalence map details of the target baseline element selected and its postconditions in a list (to see the ID, Name and description one postcondition must be selected).

- The right zone presents information about the source project. In the top part, the URL of the source assurance project can be found. Below, a tree shows the contents of the source baseline, where we can select one of the items to display its compliance map information and the contents of the source evidence model in the tree at the bottom.

![Cross-domain window](image)

**Figure 39. Cross-domain window**

The user must follow the following steps:

- Choose the source project of reuse using the “Search button”, so that the source baseline and evidence model tree will be loaded.

- Select the equivalence model and the equivalence group.

- Select the target base element that will receive the evidences to be reused, so that its compliance and equivalence map information will be loaded (highlighting in green its target elements in the trees).

- Finally, select the target Artefact and press the “Reuse” button to start the copy of the checked source Artefacts to the target selected Artefact (only one can be selected).

The source repository configuration information inside the Artefact Model Object, the Resource objects of the checked source Artefacts and the repository files related to these resources will be copied to the target evidence model. Additionally, the postconditions will be selected in the target baseline model.
2.2.7 Product/Process/Assurance Case Line Specification

To manage families/lines, it is necessary to have at disposal modelling means for systematising commonalities and variabilities. These means might be provided either as a specific solution targeting a single type of family (e.g., a process line) or as an orthogonal solution applicable to any type of family.

3.1.2.7 Variability management support at process level

This chapter describes the implementation for requirement WP6_PPA_001, described in the table above. The safety-oriented processes tend to be reused, modified and extended to individual projects in a similar manner to the product lines. However, to be able to establish the Safety-oriented Process Lines (SoPLs), the seamless integration between process engineering and variability management is required. This might be done in two possible ways: either the support for variability modelling and management is incorporated and implemented in a process engineering framework, or otherwise the integration with variability management solution needs to be achieved. This chapter focuses on the seamless integration between Eclipse Process Framework (EPF) Composer and Base Variability Resolution (BVR) Tool. The former supports the major parts of the OMG's Software & Systems Process Engineering Metamodell (SPEM) Version 2.0, while the latter is a simplification and enhancement of the OMG's revised submission of Common Variability Language (CVL). EPF Composer and BVR Tool are implemented as Eclipse plugins, which are licensed under the Eclipse Public License (EPL) Version 1.0. In a tool paper, the integration between EPF Composer and BVR Tool is discussed [35].

EPF Composer is the only available implementation of OMG's SPEM 2.0, but the migration of EPF Composer to newer versions of technologies was never performed. Accordingly, we evolved the EPF Composer from Eclipse Galileo 3.5.2 to Eclipse Neon 4.6.3 after 11 years. This is done for integration in the AMASS platform. This contribution is acknowledged by the IBM, and therefore committer status is assigned for the project. The migration is performed in four steps [36]:

1. Compatible versions of required software’s are installed from the Neon software repository and then deprecations in the source code are analysed and fixed.
2. Scheduling conflicts are resolved for the persistence of method elements (i.e., method configurations, method plugins, method content descriptions and processes) in their own folders and XMI files.
3. Appearance and height problems are resolved for the combo box which supports users in selecting the currently used method configuration, the blank views are removed from the authoring and browsing perspectives, and problems with the rich text editor are resolved for enabling users to format and style text.
4. Incompatible bundles are removed from the feature plugins, replacing bundles are added and other missing dependencies for the bundles are resolved for exporting the application. As per recommendation, the EPF Composer might be launched as a standalone application, but also in the Eclipse Integrated Development Environment (IDE).

BVR defines variability orthogonally for any MOF-compliant model, but the integration with EPF Composer brings additional challenges. EPF Composer is based on the UMA metamodel. It persists the method library contents in their own folders and files. The XMI files produced by EPF Composer are neither directly opened nor mapped at the realization editor. Therefore, the problems in XMI files have to be resolved for variability management with the BVR Tool. EPF Composer is based on the UMA metamodel. It persists the method library contents in their own folders and XMI files, in particular, method plugins, processes, content descriptions and configurations. In case of a new plugin, a plugin.xmi file is created in the new plugin directory and the reference of plugin is added to the library.xmi file. When a new capability pattern or delivery process is created, the model.xmi and content.xmi files are created in a new directory, and the reference of new process is added to the plugin.xmi file. Similarly, moving a content element to another
plugin changes plugin.xmi in both plugins. The configuration file is used to specify the working set: It records the references to included content packages and processes.

It is decided to copy the method library before resolving the problems in XMI files for two reasons. Firstly, the library might be keep running in EPF Composer. Secondly, reverting back is just required for configured processes. The packages and resource factories are registered for the UMA metamodel. Otherwise, the package and class not found exceptions would be raised. We have identified that the hypertext references (hrefs) in XMI files are based on the globally unique identifiers (GUIDs) for example uma://_ErexoKA4EeaPp8nsuu2eeew. This is the case with multiple UMA metamodel elements, such as tasks, roles, work products, tool mentors and method packages. As a result, the malformed URL exceptions are produced. The platform specific paths or otherwise Uniform Resource Identifiers (URIs) should be used instead. The support for the identification and resolution of problems with hrefs has been implemented.

Figure 40. The achievement of error free models

To create the placement and replacement fragments, the model elements are dragged and dropped to the realisation editor, and then create placement or create replacement option is selected from the context menu. This, however, produces the illegal operation exception for UMA compliant models. The analysis reveals that multiple metamodel elements have associated description implementations, such as deliverables and break down elements. Their naming structure “parent name, parent GUID” is not allowed. Accordingly, we performed temporary adaptations for supporting placements and replacements in the realisation editor. The problems are resolved for all XMI files; the method library, configurations, plugins, processes and content descriptions might be considered for variability management with the BVR Tool. The visual support for highlighting objects placements in red while replacements in blue colours, as well as retrieving selections are supported for UMA compliant models.

We have implemented a dialogue wizard to support the mapping of target configurations at the realisation editor, as shown in Figure 40 the recent/default path choice is automatically filled in the path text box otherwise the path containing a specific method library might be browsed. The dialogue wizard performs two tasks: (i) imports the contents of the method library in the target directory; and (ii) resolves problems with the XMI files. The error free models are made available in the project folder. All the model files can be opened, for example, method configurations, method plugins, method content descriptions and processes. The generation of target configurations for a software process modelled in EPF Composer is performed with VSpec, Resolution, and Realisation editors, as illustrated in Figure 41.

The generated process models are automatically exported back to the EPF Composer. If EPF Composer is running, the dialogue window pops up to inform that “the files have been changed on the file system. Do
you want to load the changes?” The selection of yes option further allows the user to save the copy of previous model. Pressing the finish button loads the derived process model in EPF Composer. It might be noted that the changes for resolving problems in XMI files and supporting the communication with realization editor had been reverted back in exported models.
Figure 41. ECSS-E-ST-40C compliant SoPL
3.1.2.8 Specification of variability at component level

This section describes the implementation for requirement WP6_PPA_004, described in the Table 2. The integration between CHESS and BVR tools represents a feasible and technically advantageous solution for variability management at product level. Similar to process lines, the generation of target product is performed with three editors: VSpec, Resolution, and Realization.

The CHESS Tool is built on top of Eclipse Papyrus. Similar to Papyrus, the CHESS model is stored in .di, .notation and .uml files. In order to visualize the diagrams, the .di file is opened with the Papyrus editor, in which the dragged model variants from model explorer or palette are dropped. As a consequence, the style information is recorded in the .notation file. The model variants, however, are stored in .uml file. The placements and replacements would have been defined for the variations in .uml file; the interactions of CHESSML compliant models with the BVR Tool are supported.

When the model variants are removed from the .uml file, the dangling stereotypes problem is caused. In particular, the repair stereotypes dialog pops up after the removal of model variants. To resolve this problem, the stereotypes applied at the placement variants are retrieved and deleted. This is done before the removal of model variants. Besides the dangling stereotypes, the orphan views appear in the diagram editors. In this context, the implemented command for clean diagrams needs to be executed; the style information of orphan views is removed from the .notation file. This is done for both opened and closed diagrams. The replacements in executed fragments have also been recorded. Specifically, the replacement variants are tracked within the .uml file and dropped at the diagram editors. After that, the Arrange All command is executed for the editors. At the opening of EPF Composer and CHESS Tool, the dialogue window pops up to inform that “the files have been changed on the file system. Do you want to load the changes?” Pressing the “Yes” button loads the tailored model.

3.1.2.9 Variability management support at assurance case level

This chapter focuses on the implementation of requirement WP6_PPA_005, described in Table 3.

To support the variability management at assurance case level, the OpenCert Tool and BVR Tool are utilised. In a similar manner to the process and product lines, the generation of target model is performed with three editors: VSpec, Resolution, and Realization. Since the BVR Tool defines variability orthogonally for any Meta-Object Facility (MOF)-compliant model (representing the Base model), the interactions of CACM compliant models with the BVR Tool are supported to map the elements of a target configuration and variability abstractions in BVR. The argumentation is stored in .arg and .arg_diagram files. The placements and replacements are defined for the variations in .arg file. For making specification intuitive and visual, the placements and replacements are highlighted in red and blue colours, respectively. The execution of configuration/resolution generates the tailored model. Accordingly, the command for propagation of injected model elements into diagram is executed.

3.1.2.10 Variability Management and Change Impact Analysis in Multiple Lines

The process, product and assurance case variability might be specified in the combined or otherwise individual models. In the combined models, the individual branches might be taken into consideration for the process and product variability. The constraints are enforced over the model elements, for which their names are considered. It is therefore important to avoid duplicates; the occurrences can also be defined. However, the BVR Tool does not support variability management and impact analysis in multiple lines.

The idea with the individual models is separation of concerns, so that the process, product and assurance case engineers work on their respective models. The interactions between process, product and assurance case models have been supported; the logical operators such as implication, alternative, negation might be used in the cross-cutting constraints. It is a meaningful way to enforce the process-product-assurance case dependencies. There is also a need to consider the occurrence specifications between the variability models of a project. The presence of elements mentioned within the constraint is first checked in the
current model. If the elements are not detected, the search is extended to other models in a project. In case the elements are detected in another model, the dialogue window pops up to inform the existence in specific model. The user, however, needs to authenticate the enforcement of cross-cutting constraints.

The resolution editor is used for specification, validation and execution of process, product and assurance case configurations. The resolutions are automatically generated from the VSpec model in which the varying choices needed to be included or excluded. It is possible to define multiple resolutions for the processes, products and assurance cases with variabilities. The constraints are used to specify the dependencies between choices. Therefore, there is a need to pair the process, product and assurance case resolutions to perform error checking and validation. The command for pairing of resolutions is incorporated; valid impact analysis and change propagation are guaranteed if the cross-cutting constraints are properly specified.

In BVR, the resolution execution was supported for a single base model. To be able to support the variant management and change impact analysis in integrated lines, the execution of two or more base models is needed, for which the source code is altered. A dialogue wizard is used to inform the possible candidates to the user, in particular, the models for which realisation fragments have been specified. The checklist selection is supported for the base models. Besides that, the user either selects the execution of cross-cutting dependencies for the purpose of impact propagation, or otherwise the whole joined resolution is executed. Accordingly, the back-propagation of tailored models is performed. The support for saving the copy of previous models is also incorporated.

3.1.2.11 Semi-automatic generation of product arguments

This section describes the implementation for requirement WP6_PPA_002, described in Table 2.

The generation of product-based arguments functionality uses the information specified in a CHESS model to generate a set of argument-fragments for each of the components from the specified model. The argument-fragments are created on the connected CDO [11] repository in the assurance case selected by the user. The generated .arg and .arg_diagram files for each component are available after generation in the corresponding “ARGUMENTATION” folder. Each argument-fragment contains information about the contracts of the corresponding component. If a contact is validated (has the status set to “validated”) the clarification of the contracts as well as supporting evidence is added to the argument. If the contract is not validated, then a challenge is added to the argument, implying that the satisfaction of the contract is challenged by the refinement results. The generator uses traceability enabled by Capra to get the traces between contracts and supporting evidence, but it also generates the traces between components, contracts and formal properties with the corresponding automatically generated argumentation elements such as claims, contexts, evidence, etc.

3.1.2.12 Semi-automatic generation of process arguments

This chapter describes the implementation for requirement WP6_PPA_003, described in the Table 2.

2.2.7.1.1 Detecting fallacies in process models

The process-based argumentations cannot ensure that the evidences are sufficient to support the claim. In order to prevent or detect fallacies in the process-based argumentations, fallacy detection plugin is developed that validates the process models and prevents the occurrence of fallacy, specifically, “Omission of Key Evidence” in process-based arguments. Omission of key evidence fallacies within the context of process argumentation are the flaws or defects in which arguments can fail to provide sufficient evidence. For example, evidences about staff competency or skills to support the process claim about designer who is responsible for the design task, which deals with the production of design-related work products are not provided. Furthermore, the tool and its related tool mentor qualification is missing in order to ensure that the evidence generated by that tool is trustworthy and valid.
Detect Fallacies plugin enables the process engineers and/or safety engineers to detect whether the process models contain all the essential information for supporting the key evidence(s). In case of omission of crucial information, recommendations are provided to aid process engineers or safety engineers in resolving deviations. Based on the recommendation(s), engineers are expected to modify the process model. The operation will only be applicable by clicking on those elements of type ProcessComponent (Capability Pattern or Delivery Process) since these elements are the ones that have all the information of process (see Figure 42).

![Figure 42. Detecting fallacies in process models menu](image)

To detect fallacies in process models and to perform the generation of fallacy free process-based argumentation, a safety process in EPF Composer is modelled according to the best practices as well as according to the standard(s). Either, the requirements and associated process life cycle can be modelled by following the IBM approach as discussed in AMASS D6.3 [30] and AMASS User Manual [18]; or the standard requirements can automatically be imported into the EPF Composer as described in AMASS D1.5 [70]. Certifications against the required standard’s requirements for the roles in process lifecycle can be described in Skills (Staffing Information) field, as shown in Figure 43. Certifications or rationales against required tool qualifications can be defined in Key considerations (Detail Information) field, as shown in Figure 44. If the evidence details are omitted or the rationale is not provided; it means that process contains the omission of key evidence fallacies.
The fallacy detecting functionality provides validation results including the list of elements containing sufficient and insufficient information (i.e., detected fallacies). In addition, the appropriate recommendations to resolve the particular deviations will be presented. These results are printed on the console (see Figure 45) as well as the validation reports, two new “.txt” files are created in the selected folder (see Figure 46). The information related to the Roles is stored under the file called “Staffing Plan Report.txt”, and the information regarding the Tools is stored in the “Tool Qualification Plan Report.txt” file. Moreover, this information is sorted in the following way: the elements with insufficient evidence appear in the first place, and then the elements having sufficient evidences are placed below.
Figure 45. Validation results printed on the console

Figure 46. Generated results reports
2.2.7.1.2 Semi-automatic generation of process arguments representing plans

The main goal of this functionality is to generate fallacy free process-based argumentation model (with diagram). A specific process plan has derived from the family of processes (managed by EPF Composer), which is automatically transformed into safety argument fragments using Epsilon Transformation Language (ETL) [8]. In Prototype P2, the generation of process-based arguments functionality is invoked from a right-click menu of ProcessComponent (Capability Pattern or Delivery Process) modelled in EPF Composer, in particular, by selecting “Transformation -> Generate Process-based Argument” option, as shown in Figure 47.

![Figure 47. Generating process-based argumentation menu (representing plan)](image)

To perform the generation of process-based arguments, the mapping between process elements (SPEM/UMA) and argumentation elements (SACM/CACM) has implemented in EPF Composer. The mapping between meta-models has also been extended. In particular, the mapping of elements is focused on the Work Breakdown Structure of processes in EPF Composer, such as Capability Pattern, Phase, Activity, Task Descriptor, Role Descriptor, Work Product Descriptor, Tool, and Tool Mentors. Since the generation of process-based argumentation focused on the planning phase of the processes, the evidences of some elements such as Work Products, Checklist, Guideline, and Example are not available in those stages of the projects. These elements have mapped into undeveloped Claim, which requires further development (see Figure 48). The evidences associated to elements have mapped into InformationElementCitation typed “solutions” showing that the particular goals have been achieved. The rationale related to element (e.g., tool qualification is not needed since source code is fully tested) is mapped into InformationElementCitation typed “justification”. The “purpose” attribute of the process is used to show that the process is compliant with the standard, which is mapped to InformationElementCitation “context”. The detailed mapping between process model, argumentation mode is provided in D6.3 [30]. The generated process-based arguments are visualized in Assurance Case Editor in OpenCert under the “ARGUMENTATION” folder. The generated elements would be visible in the editing window of the Argumentation Diagram (.arg_diagram) by using drag and drop functionality. Furthermore, the argumentation model and diagram will be stored locally in a new project into the current Workspace under the name “Argumentation”.

H2020-JTI-ECSEL-2015 # 692474
2.2.8 Reuse Discovery

This chapter describes the implementation for requirements WP6_RA_002 and WP6_RA_006, described in Table 2. This requirement has been developed in two different complementary implementations, so in the following chapter both implementations will be described.

3.1.2.13 Reuse Discovery based on OSLC-KM

The implementation of the indexing and retrieval algorithms for Reuse Discovery, based on OSLC-KM, has been an iterative process within the AMASS platform. For P1, the development and integration focused on physical SysML Papyrus Models (see chapter 2.2.8.1.1), and for P2 the focus was set on the integration with the CDO Repository to index Requirements and Evidence Models to be reused later on in the Reuse Assistant Component (see chapter 2.2.8.1.2).

2.2.8.1.1 Indexing and Retrieval of SysML Physical models

The main goal of this functionality is to allow the user to search for similar artefacts (e.g. Papyrus SysML models) within a set of indexed artefacts. This functionality was implemented for Prototype P1, and it’s a very basic connector for indexing and searching for files, based on the OSLC-KM standard [28] which allows exchanging artefacts content from different and heterogeneous sources. For the Prototype P2, this will be the starting point for the further integration with the CDO.

The core functionality is developed in KM, as part of the TRC toolset. It allows indexing the content of a SysML model in a SKB (System Knowledge Base), as shown in Figure 49, so that this SKB can receive input queries in order to get similar artefacts by a given input artefact (see Figure 54).
It was also integrated within the AMASS platform as a technological demonstrator of the OSLC-KM capabilities in order to integrate this core functionality in P1 (see Figure 51). So, this functionality is available from P1.
Figure 51. Reuse Discovery operations integrated in AMASS

The previous integration (P1) included two basic operations:

1. **Index file**: By selecting local files, it is possible to create a wide SKB to perform searches.

2. **Look for similar models**: By selecting a local file as an input, it is possible to query the SKB in order to get similar models. This search is based on the RSHP model [29], and its goal is to look for similar artefacts by artefact content (relations between elements and meta-properties).

Both operations will ask the user for a Papyrus model to work with. So far, this implementation for P1 is limited to work only with Papyrus files but not yet with other kind of SysML models (MagicDraw, Rhapsody, etc.). A further integration for P2, described in the next chapter, allows the user to select items from the CDO repository.

### 2.2.8.1.2 Indexing and Retrieval of CDO objects

The functionality developed for Prototype P2 includes the integration with the CDO database. Thus, the OSLC-KM based functionality for indexing and retrieval has been extended, so that is possible to navigate through the CDO database (manually, project by project) to index the different projects that we have. After this, a “Search” functionality is available in the Reuse Assistant component. This way, it’s possible to query for Artefacts that contain requirements of a given “SIL” level, or simply query by free text.

Figure 52. Operations integrated in the CDO Repository

Regarding the indexing process, we index two types of items:

- **Assurance Projects**: the process creates a mapping between the SIL Level of the requirements inside, and the Target Artefacts pointed by the Compliance Maps.

- **Evidence Models**: the process indexes all the active Artefact Models inside, together with the SIL Level mapping calculated by means of indexing the assurance project. This way is possible to reuse Artefacts via OSLC-KM later.
As an example, from the following project in the CDO repository, the Prototype P2 allows indexing the content via OSLC-KM through the top menu:

![Figure 53. CDO metamodel to index](image)

**Figure 53.** CDO metamodel to index

![Figure 54. Actions within the AMASS Platform](image)

**Figure 54.** Actions within the AMASS Platform

![Figure 55. Project Folder selection](image)

**Figure 55.** Project Folder selection
Once the project is selected, the content of the project in the CDO (see image) will be indexed in the OSLC-KM representation schema.

![Content of the selected project](image1)

**Figure 56. Content of the selected project**

On the other hand, the OSLC-KM service also provides with an API function to search by text. We can query SIL levels to check that the service retrieves indexed artefacts. This functionality will be integrated in the Reuse Assistant component.

![Project indexed in OSLC-KM](image2)

**Figure 57. Project indexed in OSLC-KM**

On the other hand, the OSLC-KM service also provides with an API function to search by text. We can query SIL levels to check that the service retrieves indexed artefacts. This functionality will be integrated in the Reuse Assistant component.

![Query results](image3)

**Figure 58. Query results**
3.1.2.14 Reuse Discovery based on ElasticSearch

With P1 a generic indexing feature based on Elasticsearch has been implemented in a first version. Indexing is possible at EMF (Eclipse Modelling Framework) level, independent on what kind of EMF storage system is used (CDO, XMI files, or any other). Objects and their attributes are indexed into any Elasticsearch server and can be visualized using Dashboard facilities as Kibana. The indexing is not automated yet but has to be triggered manually on an object, a resource or a whole object tree. The conceptual approach, challenges and way forward were described in detail in D6.3 [30].

With P2, an additional high-level “Search and Query API” was added to support value added features as reuse assistant or impact analysis. The API is again on rather high EMF level, that means any query can be made on high abstraction level (aka object level), is transformed into low level Elasticsearch REST calls and the reply (basically hits) is transferred back to EMF Object level. That includes resolution strategies and object identification for CDO but also file based (workspace) EMF storages.

The component requires a running Elasticsearch server. That server may run on any node in the IP network. Note that originally an Elasticsearch server could be embedded into any Java runtime environment (e.g. Eclipse Runtime) but that option was removed by the Elasticsearch development team later on. Anyway, installation and setup is rather easy and straight on all major operating systems.

ZIP packages are available at [https://www.elastic.co/products/elasticsearch](https://www.elastic.co/products/elasticsearch) and can be downloaded freely. They just need to be extracted to an arbitrary location and the server can be started by using MS CMD or UNIX batch scripts. P1 was basically built on and thus supporting Elasticsearch version 5.0.0. For P2 the code was ported to Elasticsearch 5.6.x which should be also the minimum at the server side. Although the REST API is rather stable over time, there are a few obstacles that may occur when using an older version. Note that upcoming version 7.x is not supported due to a major change in the handling of indexes and types. Version 6.x on the other hand should be supported because it offers backward compatibility.
Once the server is up and running, the AMASS client can be configured in the Eclipse preferences. By default, Elasticsearch is started on port 9200 which would lead to “http://localhost:9200” as the default server URL - assuming Elasticsearch is running on local machine. An index name must be specified as well but an arbitrary alphanumeric name can be chosen here.

![ElasticSearch Preferences in P1](image)

**Figure 60.** ElasticSearch Preferences in P1

With P2 the preferences were improved to allow to ping the server but also to clean the remote index (this was only possible using the Kibana dashboard in P1).

![ElasticSearch Preferences improved in P2](image)

**Figure 61.** ElasticSearch Preferences improved in P2
An additional toolbar shows the status of the Elasticsearch configuration and connection at any time to the user. There are two “bulbs” to reflect the status, one showing the server connection and details, the second the status of the index. Both should be green in best case. Note that the search index is created on first usage, so it will switch from yellow to green at that time.

![ElasticSearch status icons](image)

**Figure 62.** ElasticSearch status icons

### 2.3 Installation and User Manuals

The steps necessary to install the AMASS Prototype P2 are exhaustively described in the AMASS User Manual [18] and will not be repeated here. That document contains all required steps and document references to set up the tools. A pre-packaged distribution will be supplied in this iteration of the AMASS platform.

In summary, the AMASS User Manual itself is the user guide of the AMASS tool prototype implementation. The users can find the installation instructions, the tool environment description, and the functionalities described in this document.
3. Implementation Description

3.1 Implemented Modules

3.1.1 Compliance Management

We have decomposed the Compliance Management module into three components: *Standards Editor*, *Process Editor* and *Compliance Editor* (see Figure 63). The *Standards Editor* tool component includes services to capture the knowledge from the standards, while the *Process Editor* tool captures information of the life-cycles that are described in the process. The focus of the third component, the *Compliance Editor* tool, is to map the information from the standards (i.e. obligations) with the information managed in the context of a given assurance project (i.e. accomplishment). We use two toolsets for compliance management: OpenCert is used for modelling standards and processes, as well as compliance maps between those models and the assurance and certification assets created in specific projects; and EPF (Eclipse Process Framework) is used for modelling processes and for modelling standards as well.

![Compliance Management Diagram](image)

*Figure 63. Tool components for Compliance Management*

3.1.2 Management of families/lines

To manage families/lines, it is necessary to have at disposal modelling means for systematising commonalities and variabilities. These means might be provided as either a specific solution targeting a single type of family (e.g., a process line) or as an orthogonal solution applicable to any type of family

3.1.2.1 Semi-automatic generation of process-based arguments

This section provides the big picture of the solution for the automatic generation of process-based arguments within AMASS (see Figure 64). The solution embraces both phases of the *certification liaison process* (which is explicitly defined within DO-178C and implicitly in place in all certification/qualification frameworks): the planning and the execution phase.
3.1.2.2 Semi-automatic generation of process-based arguments representing plans

This section provides the big picture of the solution for the generation of fallacy free process-based arguments during the planning phase within AMASS (see Figure 65). The solution embraces, at the planning phase, a specific process plan is derived from the family of processes (managed by EPF-C & BVR Tool), then an argument is automatically generated by following MDSafeCer [32] principles and visualized via the Assurance Case Editor. Furthermore, the detect fallacies solution validates the process models (managed by EPF Composer), and prevents the occurrence of fallacy, specifically, omission of key evidence in process-based arguments.

3.1.2.3 Semi-automatic generation of product-based arguments

This section provides the big picture of the solution for the automatic generation of product-based arguments (see Figure 66). The generator uses a pre-existing argument pattern for the generation. The generated argument-fragments include only those contracts whose assumptions are validated, hence only those artefacts related to the validated contracts.
3.1.3 Cross/Intra Domain Reuse

The Cross/Intra domain reuse module is composed of two main components: *Reuse Assistance* and *Line Specification* (see Figure 67). The *Reuse Assistance* tool focuses in intra and cross-domain reuse of assurance and certification assets. The *Line Specification* tool is a composite component constituted of a seamless integrator and a set of editors enabling the management of variability. More specifically, the Line Specification can get in input models regarding Product/Process/Assurance Cases and via the Seamless Integrator these models can be cleaned if necessary and used to feed the Variability Editor, the Resolution Editor and the Realization Editor, which can be used to change the models in accordance to the Line constraints.

3.2 Source Code Description for the AMASS Tool Platform

The source code of the WP6-related components in AMASS prototype P2 can be found in the source code GIT repository [17].
The code for the Compliance Management module in Prototype Core was stored together with the code of the other building blocks in the SVN repository under “tag” to distinguish the state of the code at the time of the integrated release.

The necessary plugins for the Standards, Projects, Compliance and Processes Management Specification are:

- **org.eclipse.opencert.apm.assuranceassets**
  In this plugin, the Assurance Assets metamodel is defined and stored, and the Java implementation classes for this model are generated.

- **org.eclipse.opencert.apm.assuranceassets.edit**
  The edit plugin includes adapters that provide a structured view and perform command-based edition of the Assurance Assets model objects.

- **org.eclipse.opencert.apm.assuranceassets.editor**
  This plugin provides the user interface to view instances of the model using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet. This editor saves the generated data in a local file.

- **org.eclipse.opencert.apm.assuranceassets.editor.dawn**
  This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated model in a database instead of a local file.

- **org.eclipse.opencert.apm.assurproj**
  In this plugin, the Assurance Project metamodel is defined and stored, and the Java implementation classes for this model are generated.

- **org.eclipse.opencert.apm.assurproj.edit**
  The edit plugin includes adapters that provide a structured view and perform command-based edition of the Assurance Project model objects. This plugin also includes the import functionality from EPF process models into OpenCert Evidence and Process models.

- **org.eclipse.opencert.apm.assurproj.editor**
  This plugin provides the user interface to view instances of the model using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet. This editor saves the generated data in a local file.

- **org.eclipse.opencert.apm.assurproj.editor.dawn**
  This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated model in a database instead of a local file.

- **org.eclipse.opencert.apm.assurproj.reuse**
  This plugin includes the reuse view that offers the reuse assistance functionalities.

- **org.eclipse.opencert.apm.assurproj.utils**
  This plugin provides additional features for to the standard CheckboxTreeViewer.

- **org.eclipse.opencert.apm.assurproj.wizards**
  This plugin provides a wizard to facilitate to the user the assurance project creation process and another wizard for adding a new baseline to an existing assurance project or updating an existing baseline of a project.

- **org.eclipse.opencert.pkm.baseline**
  In this plugin, the Baseline definition metamodel is defined and stored, and the Java implementation classes for this model are generated.

- **org.eclipse.opencert.pkm.baseline.edit**
  This plugin contains a provider to display Baseline definition models in a user interface. This plugin also contains the window to view the existing compliance maps of a project and the window to create or update one project’s compliance maps.

- **org.eclipse.opencert.pkm.baseline.editor**
This plugin provides the user interface to view instances of the model in a tree-based way using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet. This editor saves the generated data in a local file.

- **org.eclipse.opencert.pkm.baseline.editor.dawn**
  This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated model in a database instead of a local file.

- **org.eclipse.opencert.pkm.baseline.diagram**
  This plugin provides the user interface to view instances of the model in a graphical way using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet. This editor saves the generated data in a local file.

- **org.eclipse.opencert.pkm.baseline.diagram.dawn**
  This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated diagram model and the standard definition model in a database instead of a local file.

- **org.eclipse.opencert.infra.mappings**
  In this plugin, the Mapping metamodel is defined and stored, and the Java implementation classes for this model are generated.

- **org.eclipse.opencert.infra.mappings.edit**
  This plugin contains a provider to display Mapping models in a user interface.

- **org.eclipse.opencert.infra.mappings.editor**
  This plugin provides the user interface to view instances of the model in a tree-based way using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet. This editor saves the generated data in a local file.

- **org.eclipse.opencert.infra.mappings.editor.dawn**
  This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated Mapping model in a database instead of a local file.

- **org.eclipse.opencert.infra.properties**
  In this plugin, the Property metamodel is defined and stored, and the Java implementation classes for this model are generated.

- **org.eclipse.opencert.infra.properties.edit**
  This plugin contains a provider to display Property models in a user interface.

- **org.eclipse.opencert.infra.properties.editor**
  This plugin provides the user interface to view instances of the model in a tree-based way using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet. This editor saves the generated data in a local file.

- **org.eclipse.opencert.infra.properties.editor.dawn**
  This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated Property model in a database instead of a local file.

- **org.eclipse.opencert.pam.procspec**
  In this plugin, the Process definition metamodel is defined and stored, and the Java implementation classes for this model are generated.

- **org.eclipse.opencert.pam.procspec.edit**
  This plugin contains a provider to display Process definition models in a user interface.

- **org.eclipse.opencert.pam.procspec.editor**
  This plugin provides the user interface to view instances of the model in a tree-based way using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet. This editor saves the generated data in a local file.
• org.eclipse.opencert.pam.procspec.editor.dawn
  This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated process model in a database instead of a local file.

• org.eclipse.opencert.pkm.refframework
  In this plugin, the Standard definition metamodel is defined and stored, and the Java implementation classes for this model are generated.

• org.eclipse.opencert.pkm.refframework.edit
  This plugin contains a provider to display standard definition models in a user interface.

• org.eclipse.opencert.pkm.refframework.editor
  This plugin provides the user interface to view instances of the model in a tree-based way using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet. This editor saves the generated data in a local file.

• org.eclipse.opencert.pkm.refframework.editor.dawn
  This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated model in a database instead of a local file.

• org.eclipse.opencert.pkm.refframework.diagram
  This plugin provides the user interface to view instances of the model in a graphical way using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet. This editor saves the generated data in a local file.

• org.eclipse.opencert.pkm.refframework.diagram.dawn
  This plugin is an extension of the previous one. It aims to communicate with the CDO Server to store the generated diagram model and the standard definition model in a database instead of a local file.

• org.eclipse.opencert.chessdiagram.adapter
  This plugin adopts CHESS editor to interact with the BVR tool bundle. BVREnabledEditor is expected to interact with the CHESSBVREditor in order to highlight/select modelling elements to be placed/replaced as well as to exportTailoredProducts.

• org.eclipse.opencert.lines (*)
  This plugin provides support for process, product and assurance case variability management, and integration of process-product-assurance case lines for the purpose of change impact analysis. The integration between EPF Composer, CHESS Tool, OpenCert Tool and BVR Tool is achieved for the establishment of lines. For the multiple/integrated lines, the cross-cutting constraints between variability models, joining of resolutions and simultaneous execution of realization fragments belonging to multiple base models are supported.

• org.eclipse.opencert.epf.generateArgumentation (*)
  This plugin provides the transformation of an EPF process (compliant with UMA metamodel) to argumentation model and diagram (compliant with CACM metamodel). The generated model and diagram are stored in the ARGUMENTATION folder of the selected assurance case project in the CDO repository. The user is prompted to select both source process model (Capability Pattern/Delivery Process) and target assurance case project. The target model (argument) is also saved locally in a new project into the current Workspace under the name Argumentation.

• org.eclipse.opencert.epf.detectFallacies (*)
  This plugin facilitates the detection of fallacies by identifying whether the safety process modelled in EFP Composer contains the sufficient information corresponding to the key evidence for supporting the specific requirement. The user is prompted to select the folder where he or she wants to store the validation reports .txt files. Moreover, the results are presented on the console.

• org.eclipse.opencert.epf.transformRequirements (*)
  This plugin provides the transformation of standard’s requirements modelled in EPF Composer (compliant with UMA metamodel) to baseline models (compliant with CACM metamodel). The generated models will be stored in selected assurance case project in the CDO repository.
• **org.eclipse.opencert.evm.evidspec.editor** (*)
This plugin provides the user interface to view instances of the evidence model in a tree-based way using several common viewers, and to add, remove, cut, copy and paste model objects, or modify the objects in a standard property sheet.
This plugin also allows the introduction of the searching parameters to find reusable artefacts by means of the OSLC-KM Reuse Discovery API or the Elastic Search Reuse Discovery API and shows the results to the user.

![Project Explorer](image.png) ![Plug-ins](image.png)

- org.eclipse.opencert.apm.assuranceassets
- org.eclipse.opencert.apm.assuranceassets.edit
- org.eclipse.opencert.apm.assuranceassets.editor
- org.eclipse.opencert.apm.assuranceassets.editor.dawn
- org.eclipse.opencert.apm.assurproj
- org.eclipse.opencert.apm.assurproj.edit
- org.eclipse.opencert.apm.assurproj.editor
- org.eclipse.opencert.apm.assurproj.editor.dawn
- org.eclipse.opencert.apm.assurproj.utils
- org.eclipse.opencert.apm.assurproj.wizards
- org.eclipse.opencert.apm.baseline
- org.eclipse.opencert.apm.baseline.dia.gram
- org.eclipse.opencert.apm.baseline.dia.gram.dawn
- org.eclipse.opencert.apm.baseline.edit
- org.eclipse.opencert.apm.baseline.editor
- org.eclipse.opencert.apm.baseline.editor.dawn
- org.eclipse.opencert.infra.mappings
- org.eclipse.opencert.infra.mappings.edit
- org.eclipse.opencert.infra.mappings.editor
- org.eclipse.opencert.infra.mappings.editor.dawn
- org.eclipse.opencert.infra.properties
- org.eclipse.opencert.infra.properties.edit
- org.eclipse.opencert.infra.properties.editor
- org.eclipse.opencert.infra.properties.editor.dawn
- org.eclipse.opencert.pam.procspec
- org.eclipse.opencert.pam.procspec.edit
- org.eclipse.opencert.pam.procspec.editor
- org.eclipse.opencert.pam.procspec.editor.dawn
- org.eclipse.opencert.pkm.refframework
- org.eclipse.opencert.pkm.refframework.dia.gram
- org.eclipse.opencert.pkm.refframework.dia.gram.dawn
- org.eclipse.opencert.pkm.refframework.edit
- org.eclipse.opencert.pkm.refframework.editor
- org.eclipse.opencert.pkm.refframework.editor.dawn
- org.eclipse.opencert.pkm.refframework.utils

**Figure 68.** Cross-Domain and Intra-Domain Reuse plugins

In addition, the following plugin from the CHESS Polarsys project has been updated:
• **org.polarsys.chess.service**
  This plugin provides some utility functions for the CHESS editor; the plugin has been extended in the context of WP6 with a new feature regarding the removal of orphan views from the Papyrus UML diagrams. Orphan views are graphical elements of a Papyrus diagram that do not have any corresponding semantic element in the UML model. For instance, orphan views can originate in a Papyrus diagram if the associated entities in the UML model are removed without using the Papyrus editor facilities; this is the case when the CHESS model is modified to implement variability at product level trough the BVR tool (see section 3.1.2.8).

  The plugin is available from [https://git.polarsys.org/c/chess/chess.git](https://git.polarsys.org/c/chess/chess.git) repository, together with the other CHESS related plugins.

Furthermore, the functionality for Reuse Discovery has been integrated within the following plugin:

• **org.eclipse.opencert.evm.oslc.km.importevid**
  This plugin provides both, the functionality for indexing Papyrus models into the KM database, through OSLC-KM, and the capability to look for similar Papyrus models given a Papyrus model as a query.

The functionality for the product-based argument generation is located in the following plugin:

• **org.eclipse.opencert.chess.argumentGenerator**
  This plugin facilitates generation of argument-fragments for each component of a CHESS model. It stores the argument files in the ARGUMENTATION folder of the selected CDO repository assurance case. The user is prompted to select both source CHESS model and target assurance case. The plugin is available on the AMASS_source.
3.3 Source Code Description for External Tools

3.3.1 OSLC-KM integration

The core functionality for the Reuse Discovery section has been developed within the TRC toolset (mainly Knowledge Manager for the WP6). The interoperability with the AMASS platform has been performed via the OSLC-KM, using some of the operations defined in the context of WP5. Specially, the implementation of the functionality of the WP6 takes advantage of the following components:

- **Rqa.Face.OslcKM**: it implements the connection of the OSLC-KM model instance with the rest of the functionalities of the TRC toolset.
- **System Representation Language (SRL)**: metamodel of the OSLC-KM layer.

3.3.2 Ontology Reuse Operations

In the context of AMASS WP6, some operations for ontology reuse have been developed in the TRC toolset (KM):

![Figure 69. Ontology Reuse Operations](image-url)
4. Conclusions

This deliverable presented the implementation work performed for Cross-Domain and Intra-Domain Reuse in AMASS Prototype P2, which is the latest version of the AMASS Tool Platform. The current support in the Platform allows a user to manage variability at the product, process and component level, semi-automatic generation of product/process arguments, and some external support provided for reuse discovery.

As its current state and prior to validation in WP2 and application in WP1, Cross-Domain and Intra-Domain Reuse support for Prototype P2 should have TRL 5 (technology validated in real use cases). Basic technological components are integrated to establish that they work together.

In addition to the implementation of further requirements in the AMASS tool Platform (and also to the general revision of some implementation for enhancement), the Prototype P2 includes the final decision upon the implementation of new features targeted at: reuse discovery, reuse assistant, variability management at the product/process/component level, and generation of product/process arguments.
## Abbreviations and Definitions

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASIL</td>
<td>Automotive Safety Integrity Level</td>
</tr>
<tr>
<td>BVR</td>
<td>Base Variability Resolution</td>
</tr>
<tr>
<td>CACM</td>
<td>Common Assurance and Certification Metamodel</td>
</tr>
<tr>
<td>CDO</td>
<td>Connected Data Objects</td>
</tr>
<tr>
<td>CHESS</td>
<td>Composition with Guarantees for High-Integrity Embedded Software Components Assembly</td>
</tr>
<tr>
<td>CHESSML</td>
<td>CHESS Modelling Language</td>
</tr>
<tr>
<td>CPF</td>
<td>Canonical Predictive Form</td>
</tr>
<tr>
<td>CPS</td>
<td>Cyber-Physical Systems</td>
</tr>
<tr>
<td>DAL</td>
<td>Development Assurance Levels</td>
</tr>
<tr>
<td>DIN</td>
<td>Deutsches Institut für Normung</td>
</tr>
<tr>
<td>EBIOS</td>
<td>Expression des Besoins et Identification des Objectifs de Sécurité (Expression of Needs and Identification of Security Objectives)</td>
</tr>
<tr>
<td>ECSEL</td>
<td>Electronic Components and Systems for European Leadership</td>
</tr>
<tr>
<td>EMF</td>
<td>Eclipse Modelling Framework</td>
</tr>
<tr>
<td>EN</td>
<td>European Norm</td>
</tr>
<tr>
<td>EPF</td>
<td>Eclipse Process Framework</td>
</tr>
<tr>
<td>ETL</td>
<td>Epsilon Transformation Language</td>
</tr>
<tr>
<td>EU</td>
<td>European Union</td>
</tr>
<tr>
<td>FT</td>
<td>Fault Tolerance</td>
</tr>
<tr>
<td>GSN</td>
<td>Goal Structured Notation</td>
</tr>
<tr>
<td>GUID</td>
<td>Globally Unique Identifier</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission</td>
</tr>
<tr>
<td>IED</td>
<td>Intelligent Electronic Devices</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IL</td>
<td>Impact Level</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent Transport Systems</td>
</tr>
<tr>
<td>JU</td>
<td>Joint Undertaking</td>
</tr>
<tr>
<td>KM</td>
<td>Knowledge Manager</td>
</tr>
<tr>
<td>OCRA</td>
<td>Othello Contracts Refinement Analysis</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>OSLC</td>
<td>Open Services for Lifecycle Collaboration</td>
</tr>
<tr>
<td>OSLC-KM</td>
<td>OSLC for Knowledge Management</td>
</tr>
<tr>
<td>PhD</td>
<td>Philosophiae Doctor (neolatin; = doctor of philosophy)</td>
</tr>
<tr>
<td>RSHP</td>
<td>Relationship Model</td>
</tr>
<tr>
<td>RTCA</td>
<td>Radio Technical Commission for Aeronautics</td>
</tr>
<tr>
<td>RT</td>
<td>Real-Time</td>
</tr>
<tr>
<td>SACM</td>
<td>Structured Assurance Case Metamodel</td>
</tr>
<tr>
<td>SDLC</td>
<td>(Microsoft) Secure Development Life Cycle</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Explanation</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>SIL</td>
<td>Safety Integrity Level</td>
</tr>
<tr>
<td>SKB</td>
<td>System Knowledge Base</td>
</tr>
<tr>
<td>SL</td>
<td>Security Level</td>
</tr>
<tr>
<td>SoPLs</td>
<td>Safety-oriented Process Lines</td>
</tr>
<tr>
<td>SPEM</td>
<td>Software &amp; Systems Process Engineering Metamodel</td>
</tr>
<tr>
<td>STO</td>
<td>Scientific and Technical Objective</td>
</tr>
<tr>
<td>SVN</td>
<td>Subversion</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>SysML</td>
<td>System Modelling Language</td>
</tr>
<tr>
<td>TOM</td>
<td>Trade-Off Method</td>
</tr>
<tr>
<td>TRC</td>
<td>The REUSE Company</td>
</tr>
<tr>
<td>TRL</td>
<td>Technology Readiness Level</td>
</tr>
<tr>
<td>UMA</td>
<td>Unified Method Architecture</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification and Validation</td>
</tr>
<tr>
<td>XMI</td>
<td>XML Metadata Interchange</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
</tbody>
</table>
References

[1] The OPENCOSS project  http://www.opencoss-project.eu/
[12] OSLC http://open-services.net/specifications/
[13] AMASS project: D2.1 Business cases and high-level requirements (28 February 2017)
[14] AMASS project: D3.6 Prototype for architecture-driven assurance (c) (31 August 2018)
[16] AMASS project: D5.3 Design of the AMASS tools and methods for seamless interoperability (b) (29 June 2018)
[20] AMASS D1.1 Case studies description and business impact (30 November 2016)
[26] OpenCert project https://www.polarsys.org/opencert/

4 The AMASS SVN code repository is open to AMASS partners with the same credentials as the SVN document repository. In case that people outside the project need access, please contact the AMASS Project Manager (alejandra.ruiz@tecnalia.com)

5 The final version of the manual will be integrated in D2.5 AMASS User guidance and methodological framework (m31).
[31] AMASS D1.5 AMASS Demonstrators (b), 21 April 2018


[33] Regorous Process Designer is available under an evaluation license http://www.regorous.com


[37] AMASS D2.5 User Guidance and Methodological Framework, October 2018.
Appendix A: Document changes with respect to D6.5

New Sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.4</td>
<td>Solution for Process Compliance Checking</td>
</tr>
<tr>
<td>3.1.2.10</td>
<td>Variability Management and Change Impact Analysis in Multiple Lines</td>
</tr>
<tr>
<td>2.2.7.1.1</td>
<td>Detecting fallacies in process models</td>
</tr>
<tr>
<td>2.2.7.1.2</td>
<td>Semi-automatic generation of process arguments representing plans</td>
</tr>
<tr>
<td>3.1.2.13</td>
<td>Reuse Discovery based on OSLC-KM</td>
</tr>
<tr>
<td>3.1.2.14</td>
<td>Reuse Discovery based on ElasticSearch</td>
</tr>
<tr>
<td>3.1.2.16</td>
<td>Semi-automatic generation of process-based arguments representing plans</td>
</tr>
<tr>
<td>3.3</td>
<td>Source Code Description for External Tools</td>
</tr>
<tr>
<td>Appendix A</td>
<td>Appendix A: Document changes with respect to D6.5</td>
</tr>
</tbody>
</table>

Modified Sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.2</td>
<td>Tailoring of Standards Models to Specific Projects</td>
<td>Completed with implementation in P2</td>
</tr>
<tr>
<td>3.1.2.4</td>
<td>Compliance Mappings in EPF</td>
<td>Added transformation of standard’s requirements modelled in EPF composer to baseline models</td>
</tr>
<tr>
<td>3.1.2.5</td>
<td>Cross-systems reuse (intra standard or intra domain product upgrade)</td>
<td>Completed with the integration of the Reuse Discovery functionalities</td>
</tr>
<tr>
<td>3.1.2.7</td>
<td>Variability management support at process level</td>
<td></td>
</tr>
<tr>
<td>3.1.2.8</td>
<td>Specification of variability at component level</td>
<td></td>
</tr>
<tr>
<td>3.1.2.9</td>
<td>Variability management support at assurance case level</td>
<td></td>
</tr>
<tr>
<td>2.2.8</td>
<td>Reuse Discovery</td>
<td>Implemented functionalities for P2 (ElasticSearch and OSLC-KM)</td>
</tr>
<tr>
<td>3.2</td>
<td>Source Code Description for the AMASS Tool Platform</td>
<td>Revision of plugins</td>
</tr>
<tr>
<td>4</td>
<td>Conclusions</td>
<td>Included requirements completeness table</td>
</tr>
<tr>
<td></td>
<td>Abbreviations and Definitions</td>
<td>Completed list</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>Completed list</td>
</tr>
</tbody>
</table>