ECSEL Research and Innovation actions (RIA)

AMASS

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

Methodological guide for cross/intra-domain reuse (b) D6.8

Work Package:	WP6: Cross-Domain and Intra-Domain Reuse
Dissemination level:	PU: Public
Status:	Final
Date:	16 November 2018
Responsible partner:	Helmut Martin (VIF)
Contact information:	Helmut.martin@v2c2.at
Document reference:	AMASS_D6.8_WP6_VIF_V1.0

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

Contributors

Names	Organisation
Helmut Martin, Bernhard Winkler, Robert Bramberger	Virtual Vehicle (VIF)
Muhammad Atif Javed, Faiz Ul Muram, Irfan Sljivo, Barbara Gallina, Julieth Castellanos	Maelardalen Hoegskola (MDH)
Jose Luis de la Vara, Miguel Rozalen, Jose María Álvarez	Universidad Carlos III de Madrid (UC3)
Luis Alonso, Borja López, Elena Gallego	The REUSE Company (TRC)
Alejandra Ruiz, Angel López, Huáscar Espinoza (partial contribution)	Tecnalia Research & Innovation (TEC)
Huáscar Espinoza (partial contribution)	Commissariat à L'énergie Atomique et aux Energies Alternatives (CEA)
Marc Sango	Alliance pour les technologies de l'informatique (A4T)
Staffan Skogby, Detlef Scholle	ALTEN Sweden (ALT)
Jan Mauersberger	medini Technologies AG (AMT)

Reviewers

Names	Organisation
Morayo Adedjouma (Peer Reviewer D6.7 and D6.8)	Commissariat à L'énergie Atomique et aux Energies Alternatives (CEA)
Stefano Tonetta (Peer Reviewer D6.8)	Fondazione Bruno Kessler (FBK)
Thomas Gruber (Peer Reviewer D6.7)	Austrian Institute of Technology GmbH (AIT)
Cristina Martínez (Quality Manager D6.7 and D6.8)	Tecnalia Research & Innovation (TEC)
Barbara Gallina (TC review D6.7 and D6.8)	Maelardalen Hoegskola (MDH)
Jose Luis de la Vara (TC review D6.7 and D6.8)	Universidad Carlos III de Madrid (UC3)
Alejandra Ruiz (TC review D6.7)	Tecnalia Research & Innovation (TEC)

TABLE OF CONTENTS

List of Figures

List of Tables

Executive Summary

This document is the second deliverable associated with the AMASS Task 6.4 “Methodological Guidance for Cross-Domain and Intra-Domain Reuse”, which provides the methodological guide for the AMASS reuse approach. The deliverable is based on the functionalities supported by the third and final prototype (P2) of the AMASS Platform.

This document focuses on methodologies and supporting tools developed in WP6. To make the document self-contained to a certain extent, background information regarding the AMASS reuse and variability concepts, embracing the three dimensions, process, product, and assurance case, is given. Additionally, the fundamental functionalities of the tools, which support the execution of the workflow, are recalled. The methodological guide provides detailed guidance about reuse assistance, management of lines, automatic generation of arguments, transformation of standards to baseline models, compliance mapping, compliance checking, reuse discovery, representation of safety standards with semantic technologies, reuse of safety and security artefacts and model-based testing.

For a more general overview and guidance for the AMASS approach, including methods and techniques provided by others WPs, the reader is referred to D2.5 [2] “AMASS User Guidance and Methodological Framework” as well as deliverables D3.8 [5], D4.8 [6] and D5.8 [7]. These deliverables provide guidance for the AMASS architecture-driven assurance, for the AMASS multi-concern assurance, and for the AMASS seamless interoperability approach, respectively. Deliverable D2.5 provides methodological guidance for specific AMASS core features, and a user manual with detailed descriptions of the AMASS tool platform specific functions and how to use them.

This deliverable represents an update of the AMASS deliverable D6.7 [14] released at January 2018 (m22). The sections modified with respect to D6.7 have been marked with an asterisk (*), and the details about the differences and modifications are provided in Appendix A.

1. Introduction

Embedded systems have significantly increased in technical complexity towards open, interconnected systems. Data exchange requires interconnection across trusted boundaries of the system. This leads to security issues like hacking and malicious attacks against interfaces, which could invalidate existing safety measures. On that way, security issues may lead to safety issues. Software updates with new features can increase safety but introduce additional challenges on cybersecurity. The rise of complex Cyber-Physical Systems (CPS) has led to many initiatives to promote reuse and automation of labour-intensive activities such as the assurance of their dependability. The AMASS project builds on the results of two large-scale projects, namely OPENCOSS [16] and SafeCer [17]. These projects dealt with reuse, assurance and certification of software-intensive critical systems using incremental and model-based approaches.

The AMASS project enhances existing approaches with multi-concern aspects like the interaction between functional safety and cybersecurity. Well-defined reuse processes can systematize the derivation of new development processes based on an existing base process. Especially commonalities like related activities, methods and argumentation fragments can be easily reused. This reuse approach decreases effort and improves the completeness and accuracy of safety- and security cases. Users must keep in mind that improved reusable activities and methods are no guarantee that they are fully correct and complete especially with regard to security.

AMASS project partners have developed tools and integrated them to the AMASS tool platform or provide the capability to interoperate with the platform. As result, the AMASS Tool Platform is capable to ensure interaction between internal and external tools. This deliverable provides a methodological guide related to reuse concepts and their implementation in specific tools of the AMASS project. It provides an overview about key concepts like reuse assistance, management of lines, automatic generation of arguments, transformation of standards to baseline models, compliance mapping, compliance checking, reuse discovery, representation of safety standards with semantic technologies, reuse of safety and security artefacts and model-based testing.

Reuse assistance functionality together with systematic variability management enable intra and cross-standard reuse of assurance and certification assets. Automatic generation of process and product-based arguments simplifies the generation of assurance cases and supports reuse. Reuse discovery provides a methodology for the identification of reusable information based on their semantics.

This methodological guide describes how to use the AMASS tools with help of examples and detailed process steps. The workflow is presented with the aid of activity diagrams or sequences of to-be-followed steps. The steps are meant to give an example of usage of the tool trying to cover all relevant features. The user is referred to the AMASS Platform User Manual [2] to get a deeper knowledge about the specific options provided by the AMASS tools.

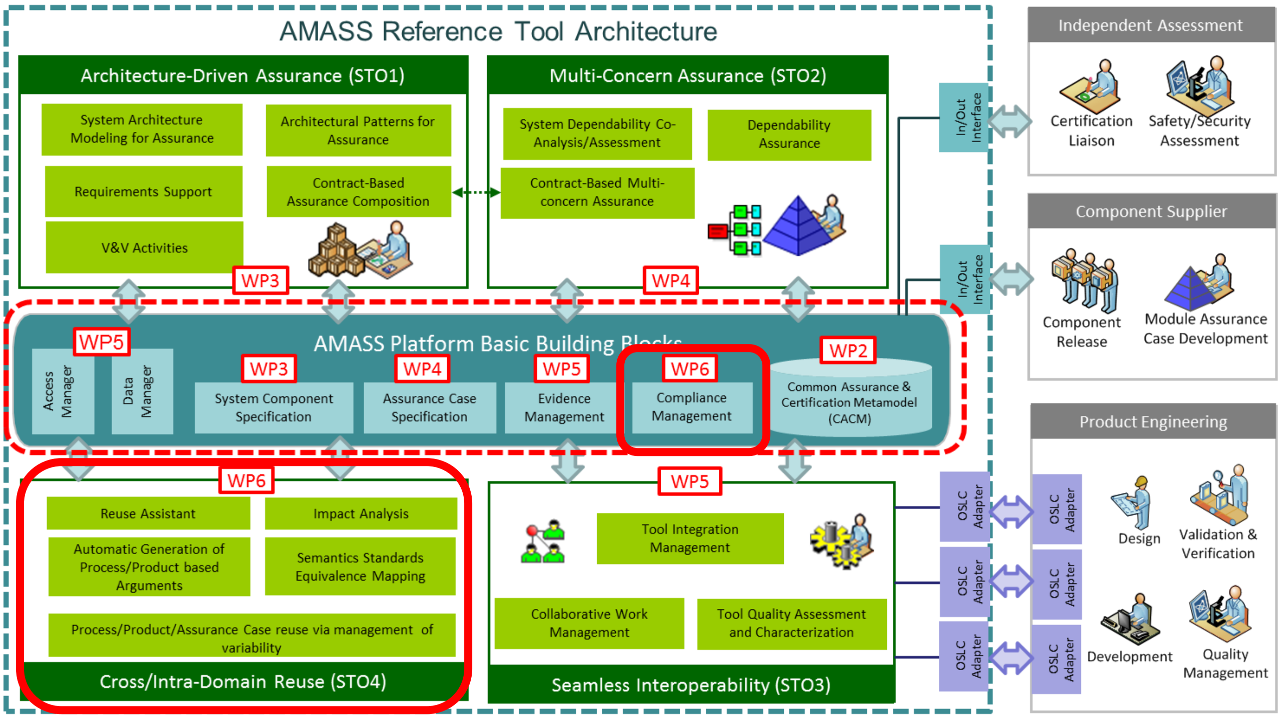

AMASS Reference Tool Architecture
Architecture-Driven Assurance (STO1)
System Architecture Modeling for Assurance
Architectural Patterns for Assurance
Requirements Support
Contract-Based Assurance Composition
V&V Activities
WP3
Multi-Concern Assurance (STO2)
System Dependability Co-Analysis/Assessment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

Figure 1. Compliance management and reuse in relation to other work packages

Figure 1 provides a general overview of the different AMASS Scientific Technical Objectives (STOs) and how they are implemented in the AMASS project by specific Work Packages (WPs). This document focuses on methodology guidelines for cross and intra-domain reuse and the related tools developed in WP6.

For a more general overview and guideline of the AMASS approach including methods and techniques provided by others WPs, the reader is referred to D2.5 “AMASS User Guidance and Methodological Framework” [2]. The deliverable D2.5 provides methodological guidance for specific AMASS core features, which are described in more detail in the deliverables D3.8 [5], D4.8 [6], D5.8 [7] and D6.8. These deliverables are the final versions of the methodological guidance deliverables based on D3.7 [4], D4.7 [9], D5.7 [10] and D6.7 [14]. They provide guidance for the AMASS approaches concerning architecture-driven assurance, multi-concern assurance seamless interoperability and cross/intra-domain reuse.

The deliverable D6.3 “Design of the AMASS Tools and Methods for Cross and Intra-Domain Reuse” [12] recalls the context and objectives of WP6 focusing on Task 6.2 “Conceptual Approach for Cross-Domain and Intra-Domain Reuse”. The conceptual approach is applied in Task T6.3 “Implementation for Cross-Domain and Intra-Domain Reuse” and leads to the deliverable D6.6 “Implementation for Cross-Domain and Intra-Domain Reuse” [13]. Task T6.4 “Methodological Guidance for Cross-Domain and Intra-Domain Reuse” has created D6.8 (the document at hand).

2. Cross-Domain and Intra-Domain Reuse Overview

2.1 Background

As it was discussed in D6.1 [11], the challenge is related to the management of reuse, to the scope of reuse, and to the reuse methodology to be applied. For better clarification of the problem, consider Figure 2 that illustrates a reuse taxonomy created to support the reader. The taxonomy is based on previous work by Ruiz in [31].

Each branch in the diagram shows a different type or aspect of reuse. There are scenarios that combine different reuse types and aspects, depending on the perspective. The **Engineering aspect** focuses on the idea of reuse of engineering practices. The **Stakeholders' aspect** focuses on different points of view and different interests depending on involved stakeholders. The **Certification data aspect** focuses on the reuse of certification related data (e.g. assurance cases) and work products (e.g. design specifications).

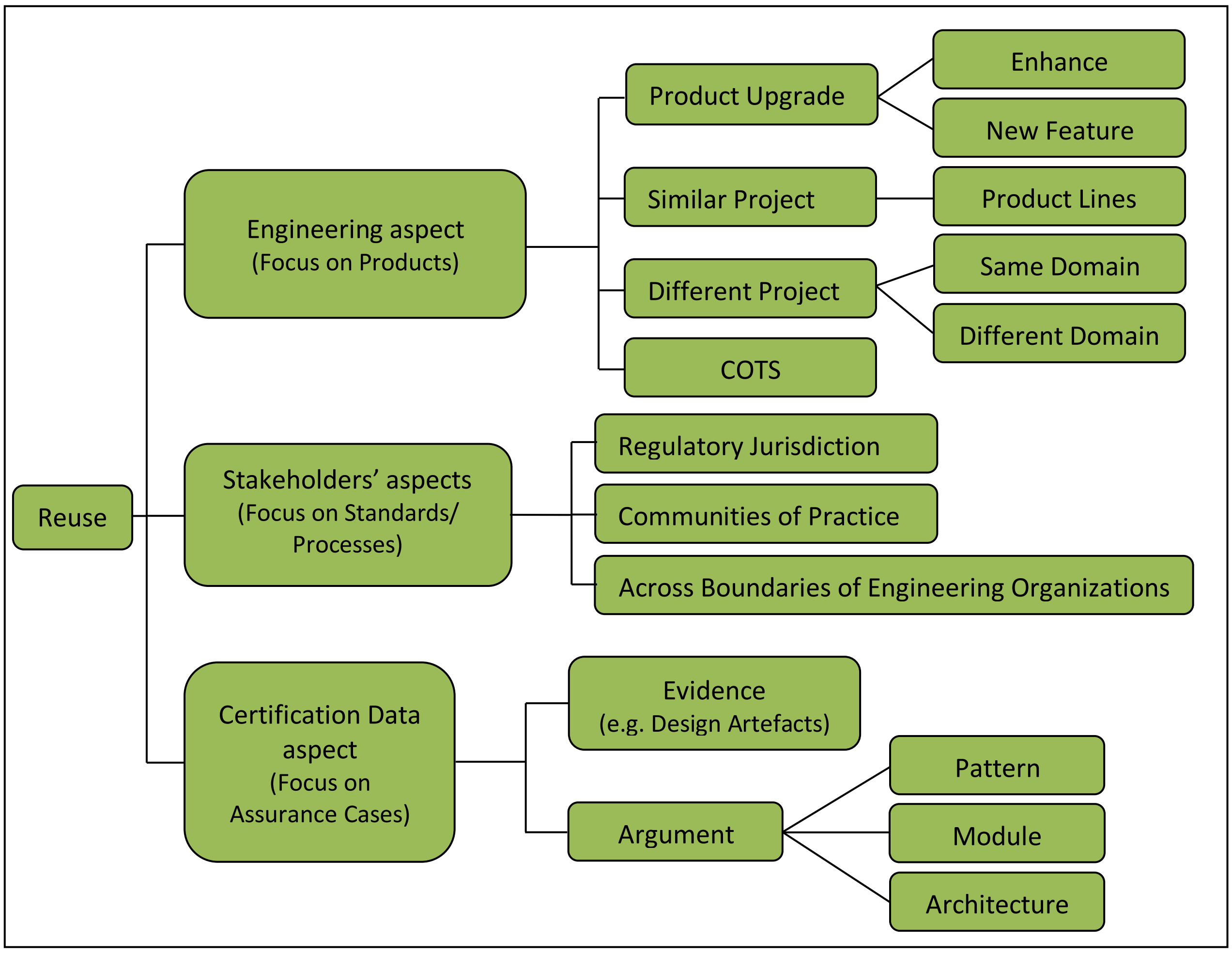

Reuse
Engineering aspect (Focus on Products)
Product Upgrade
Enhance
New Feature
Similar Project
Product Lines
Different Project
Same Domain
Different Domain
COTS
Stakeholders' aspects (Focus on Standards/ Processes)
Regulatory Jurisdiction
Communities of Practice
Across Boundaries of Engineering Organizations
Certification Data aspect (Focus on Assurance Cases)
Evidence (e.g. Design Artefacts)
Argument
Pattern
Module
Architecture

Figure 2. Reuse Taxonomy in the scope of AMASS

All the above-listed reuse types are in the scope of the AMASS project.

Product, process and assurance case are strongly related. A variation in the product has ripple effects on the process and the assurance case. To exemplify these dependencies at a higher level, Figure 3 illustrates a 3D space populated by families of items. These families represent assurance cases, processes and product-related artefacts.

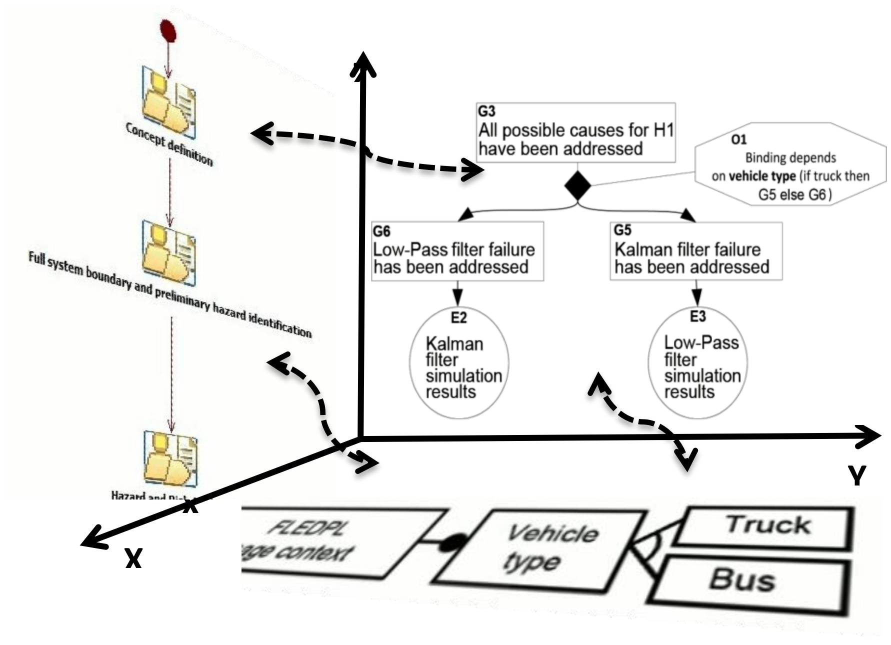

G3
All possible causes for H1 have been addressed
O1
Binding depends on vehicle type (if truck then G5 else G6)
G6
Low-Pass filter failure has been addressed
G5
Kalman filter failure has been addressed
E2
Kalman filter simulation results
E3
Low-Pass filter simulation results
Concept definition
Full system boundary and preliminary hazard identification
Y
X
Vehicle type
Truck
Bus

Figure 3. Dependencies between processes, products, and assurance cases [33]

A detailed description of reuse scenarios and reuse dimensions is available in deliverable D6.1 "Baseline and requirements for cross/intra-domain reuse" [11] and a detailed design of the reuse dimensions and their dependencies is available in D6.3 [12].

2.1.1 Process-related reuse

Variability factors in development processes include, among others, complexity, project and team size. In dependable systems engineering, stringency, dependent on the criticality of the to-be-developed system, is an additional variability factor. This factor is constrained by the requirements, imposed by the increasing amount of multi-concern normative documents. In such context, process variability management is becoming a strategic ability. To systematically manage all those variability factors, it is useful to combine: the notion of Safety-oriented Process Line Engineering (SoPLE) [32], which extends process-lines engineering in the context of dependable engineering; the Eclipse Process Framework Composer (EPF-C) [24], which offers a powerful solution to engineer processes; and the Base Variability Resolution Tool (BVR-T) [36], which offers a powerful solution to manage variability. This combination presents numerous challenges, which have been faced to obtain a seamless integration of EPF-C and BVR-T, delivered as plugin, and offer a technical solution for SoPLE.

2.1.2 Product-related reuse

Product (e.g., software) reuse can be defined as the use of an existing product or knowledge about that product to construct new products. Its main purpose is to improve both the quality and the productivity of new products. In systems that require high quality and integrity assurance, e.g., safety critical systems, reuse of the product itself is not sufficient without the reuse of the knowledge of the product [45]. The product and the knowledge of the product are subject to reuse and are often referred to as reusable assets.

Whether reuse is planned (systematic) or "ad hoc" (non-systematic) has significant influence on the quality and integrity of the system within which reuse is performed. Non-systematic reuse has been shown to be hazardous in safety-critical systems, hence systematic approaches are usually required to build a high-integrity system from reusable assets. Systematic reuse approaches pay special attention to the context and environment of the asset being reused so that it can be checked whether the asset is valid for reuse or not. Some of the systematic approaches to reuse that have emerged over years include Product-line Engineering (PLE) and Component-based Software Engineering (CBSE). The main goal of CBSE is to support development of systems as composition of reusable components, while PLE is a planned reuse approach that focuses on achieving reuse within a family of products.

2.1.3 Assurance case-related reuse

Assurance cases can be defined as a set of auditable claims, arguments, and evidence created to support the claim that a defined system/service will satisfy some particular requirements [30]. When applying reuse approaches to assurance cases, two main concepts are used: argument patterns and argument modules.

Assurance case patterns are considered to be one of the main approaches for managing reuse of safety assurance. A safety case pattern provides a mean of explicitly and clearly documenting common elements found in safety cases, and it also promotes the reuse of best practices for safety assurance. They are highly used to reuse the best practices in argumentation. Different works have been done in relation to collect the more adequate patterns together with guidelines for instantiation [40]. Argument fragments can be considered as a type of argument pattern. They are part of an assurance case, but they cannot be read isolated, they should be included with the rest of the assurance case.

A similar element but not equal is the assurance case module. A safety case module may correspond, among other things, to an interrelated set of safety engineering activities, scope of responsibilities of a particular engineering organisation, well-defined sub-system or equipment used within the overall safety-critical platform. An assurance case module does not need to be instantiated. When it is created, reuse should be already taken into account. It should clearly identify the assumptions (away goals, away context, away solutions) and guarantees (public claims) that it expects to share with other modules in the assurance case. The assumptions and guarantees of the modules are validated in argument contracts. In this case, the argument fragment can be understood in isolation and it is a piece of reusable argument that provides more and consistent arguments when included with the rest of the assurance case. [41]

Both topics have been widely explained and discussed in the context of WP4 and its deliverables.

2.1.4 Compliance management

To make the deliverable self-contained, in this subsection, the AMASS vision for compliance management developed in D6.3 [12] is recalled. Compliance management deals with the provision of evidence and justification regarding conformance to requirements coming from the standards. For instance, a development plan represents the evidence that a plan has been conceived and documented in compliance with the requirements. To ease the communication between the applicant and the certification body, the evidence alone is not enough. A justification in terms of either a checklist (concise compliance report given in a tabular format), an argument, or some proof (e.g. a verification report) should also be provided to show/argue/prove that the plans comply with the requirements.

To manage compliance, it is necessary to properly interpret and model the requirements coming from the standards, to provide the evidence expected to substantiate such requirements, and to show the relation between requirements and evidence. The vision of AMASS for compliance management is exemplified in Figure 4. Compliance management concerns the activities aimed at fulfilling the normative requirements and in charge of delivering justifications of compliance, where a justification of compliance may take different forms: a mapping table indicating which process elements act as evidence of the satisfaction of the requirements coming from the standards; an argument explaining why certain evidence is linked to certain requirements; a formal proof proving that a certain process trace satisfies a certain set of formalized requirements; or an ontology linking together standard-related concepts with process-related concepts.

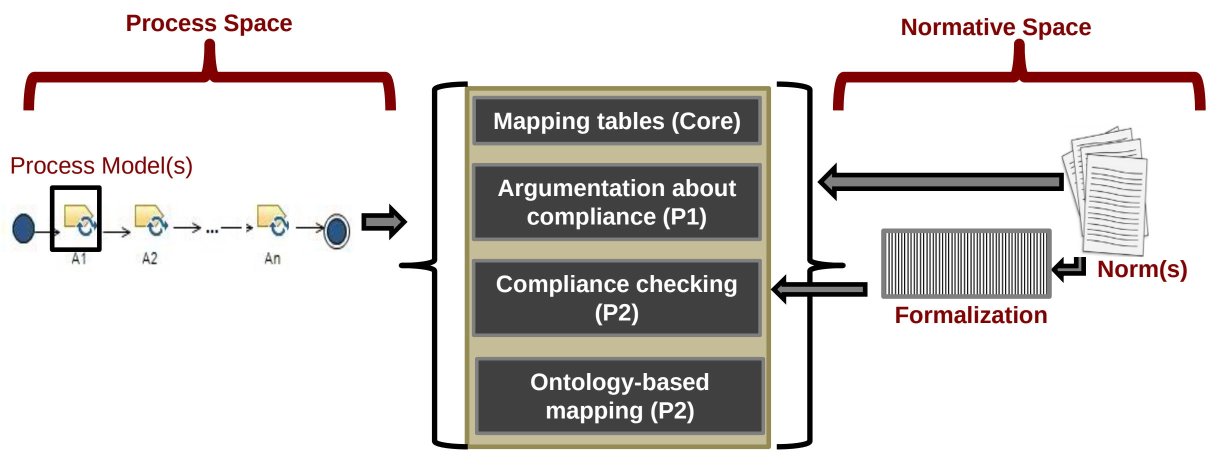


Figure 4. Compliance Management Vision

2.1.5 Reuse of compliance accomplishments

As it was mentioned in D6.1 [11], a principal challenge for AMASS is that currently the existing standards across the avionics, automation, space, railway and automotive domains exhibit wide variation, partial inconsistency, and semantic discrepancy in terms of treatment of common assurance and certification concepts. Consequently, the provision of a means for the reuse of certification artefacts in the following target circumstances might be hindered:

- ***Recertification-Reuse of approvals from one application domain to another:*** Reuse of approvals (certifications, assessments, evidence, etc.) in order to reduce time/efforts when the user wants to certify/assess a product with respect to a domain-specific standard that has already been approved in an application domain with respect to a standard from another domain.
- ***Reuse of approvals within a domain:*** Reuse of approvals (certifications, assessments, evidence, etc.) in order to reduce time/efforts when the user wants to certify/assess a product with respect to a domain-specific standard that has already been approved in the same application domain with respect to another standard [version] from the same domain, but with a different scope or corresponding to another version of a standard.

  Note that, at a first glance, this last circumstance might be considered remote, contingent. Moreover, since transitional arrangements are usually specified by the new standard/update, this condition might even be considered easier to tackle with respect to the former. Recent experience in the automotive domain shows that transitional arrangements are not always at disposal and thus the introduction of a new standard within the same domain can represent a true challenge.

The engineer needs to interpret the requirements and objectives of the standards, which will apply to the specific situation, and sometimes this is open to interpretations.

Beyond the interpretation of standards, when trying to analyse the different standards, it is beneficial to create a common framework so that the peculiarities and commonalities of the standards can be highlighted. AMASS focuses on creating such a common framework in order to compare standards from various industrial domains (e.g., avionics, automotive, railway and medical devices). These approaches facilitate the reuse of process-related assets.

2.1.6 Reuse and component-based system engineering (*)

The main goal of Component-Based System Engineering (CBSE) is to support development of systems as composition of reusable components. With complex systems this brings the challenges of combining newly developed components with reused components.

The processes and set of tools supporting reuse must adhere to standards in the domain. The functional safety standards (IEC 61508 [62], ISO 26262 [20] and others) for the different domains of cyber-physical systems specify verification artefacts. The reuse of components can improve if the integration and the V&V-activities in the development cycle are managed in line based on domain specific standards. The major challenge to reuse is the integration and verification of multi-concern cyber-physical systems with functional safety standards.

Major challenges of short Time To Market (TTM) are generated from the growing complexity in combined industrial cyber-physical systems. If the concept of reuse is managed through the product life cycle, these challenges can be met. The life cycle challenges must also be addressed with improvements in the V&V process and in the tool chain.

Following the safety standards, the artifacts must be supported through the life cycle of the products. The produced artifacts in the analysis and design phases from requirements, safety cases, models, specifications, and program code must follow through the different test phases including verification and validation.

To enable more reuse of components, new methods and tools are introduced. To increase the reuse in CPS development with multi-concern, new methods and tools should be used to automatically generate the artefacts for verification with automatic test case generators for verification. The automation and verification will also need patterns for the artefacts with model-based testing (MBT) technologies. To reuse components there is a need for integration patterns in the combination of reuse and new designed components. The measures are in the system integration, verification and validation phases of the system development cycle. The patterns for this are further described in section 3.11.

To enable reuse of components, the set of tools in a project must support the flow of designed models, code and test cases and other information elements that needs to “flow forward” to the development cycle. To enable a more efficient development, the elements in the development cycle must support the exchange of artefacts through the development cycle. The tool chain that generates those artifacts must be able to exchange the artefacts in the form of system models, test cases, etc. in defined format. The artefacts are exchanged through Open System for Life cycle Collaboration (OSLC) or other means.

The required artefacts for system integration, verification and validation must be prepared when the reused components are introduced in the development cycle for the safety critical systems. To efficiently produce these artefacts, the tool chain should the able exchange the necessary artefacts of integration and V&V. The safety standards have also a requirement on storing of the artefacts for the product life cycle from the V&V result.

The work flow needs to be iterated through models and test cases as they are cross-checked by automated test case generator tools. The pattern for this will be described in the section 3.1.1 on model-based test tools.

2.2 Cross-Domain and Intra-Domain Reuse

2.2.1 Main Idea

Intra domain reuse of CPS or parts of a system is challenging because products are qualified or certified for a specific application in a specific context. Increased connectivity makes reuse difficult because of easy occurrence of unintended behaviour. The main challenges, which the user is faced with when reusing, are mentioned in D6.1 [11]. Challenges in cross domain reuse emerge because different domains demand different standards. If a product should be used in a different domain, the worst case is that a full certification process must be applied again, because without a fitting reuse strategy the product has to be considered as totally new. The try to reuse CPS in other projects causes that assumptions and constraints may change (e.g. environment). Results from previous projects show that actual domain specific standards are not helpful in many cases, because they often do not support reuse activities and furthermore, they hardly deal with functional safety and cybersecurity from the co-engineering point of view. To resolve the issue, WP6 addresses these challenges related to cross- and intra-domain reuse as well as cross concern reuse. In addition, safety and security co-engineering is considered as a joint activity.

Approaches from prior projects (e.g. SafeCer [17] and OPENCOSS [16]) have been refined and further developed. During the project reuse methodologies for product-, process- and assurance case reuse have been elaborated. D6.8 provides detailed information about how to use the reuse-enabler tool support and how to apply the underlying approach.

2.2.2 Tool Support Overview (*)

This section gives a short overview about the tools which are part of the AMASS platform or connected to it. These tools are EPF-Composer, BVR Tool, OpenCert, Knowledge Manager, Safety Architect, Elasticsearch & Kibana, and Regorous.

Eclipse Process Framework Composer

EPF-C provides support for authoring, tailoring and deploying (software) systems development processes. This means that process structures containing all necessary process elements (e.g., activities, tasks, roles, work products, etc.) can be specified. EPF-C is based on the Unified Method Architecture (UMA) metamodel, which is an evolution of the OMG's SPEM 1.1 [28]. The major parts of UMA are incorporated in SPEM 2.0 [29]. Although EPF-C has attracted considerable attention in both academia and industry, its development environment is based on the previous technology versions, for example Eclipse Galileo, JRE 1.5, etc. The migration of EPF-C is critical for allowing the integration with other tools and applications in the Eclipse Platform. Accordingly, EPF-C has been evolved from Eclipse Galileo 3.5.2 to Eclipse Neon 4.6.3. The migration of EPF-C has involved the resolution of unresolved/outdated dependencies, problems with the storage of libraries, method contents, processes and configurations, appearance of combo-box for selection of configuration, and generation of standalone application. EPF-C might also be launched within the Eclipse IDE and is integrated in the AMASS Platform.

CHESS

CHESS [44][67] provides a model-driven, component-based methodology and tool support for development of safety-critical systems. Its underlying modelling language (CHESSML) is implemented as Unified Modelling Language (UML), Systems Modelling Language (SysML) and Modelling and Analysis of Real Time and Embedded systems (MARTE) profile. In particular, UML Version 2.5 is utilized for modelling the software aspects, SysML Version 1.1 for modelling the requirements and MARTE Version 1.1 for describing the real-time aspects. The CHESS design space is composed of particular elements: (i) data types, events and interfaces; (ii) component types; (iii) component implementations; (iv) component instances; (v) component bindings; (vi) hardware topology description and target platform elements. The remaining elements concerns the real-time architecture: (vii) containers; and (viii) connectors.

The CHESS Tool is made available as Eclipse Polarsys project. It is built on top of Eclipse Papyrus to enforce working with different views: requirement, system, component, deployment, analysis and PSM views. Therefore, the CHESS Tool supports throughout the development process, from the requirements, to the system’s architecture, down to the software design, and its deployment to hardware components. Besides that, the dependability (failure logic and state-based) analyses and code generation for Ada language are supported.

BVR Tool

BVR [35][36] is a language built on top of CVL [38] for enabling variability modelling in the context of safety-critical systems engineering. The language is implemented as a series of Eclipse plugins, which support feature modelling, resolution, realization and derivation of specific family members, as well as their testing and analysis. The overall implementation is called BVR Tool and has been integrated into the AMASS

Platform. Because the language defines variability orthogonally for any Meta-Object Facility (MOF)-compliant model (representing the Base model), communication with other tools is needed to map the elements of a target configuration and variability abstractions in BVR. The generation of target configurations is performed with three editors: VSpec, Resolution, and Realization. The *VSpec* editor permits variability engineers to create VSpec models, which are an evolution of the Feature-Oriented Domain Analysis (FODA) [39], usually called Feature Models. The *Resolution* editor permits variability engineers to perform the resolution and obtain resolved models. The *Realization* editor is based on the placements and replacements within the fragment substitutions. Fragment substitution removes elements within the placement and substitutes them with the elements in replacement.

OpenCert

OpenCert is a product and process assurance management system. It is part of the AMASS Platform to support the compliance assessment and certification of safety-critical systems in sectors such as aerospace, railway and automotive. OpenCert helps create a transparent view of the process and product quality against a set of harmonized compliance requirements derived from standards and regulations. Using knowledge-based systems, quantitative methods and modular reuse techniques, OpenCert reduces compliance management and (re-)certification costs.

OpenCert features to support compliance management and reuse are:

- Standards Specification
    - Capture information from standards.
    - Map equivalences between different standard assets (artefacts, activities, requirements, techniques).
- Assurance Project Management
    - Create assurance projects.
    - Define assurance project baseline (project plan with standard requirements).
    - Define access permission for users.
- Assurance Case Management
    - Define modular assurance structure.
    - Develop claims and links to evidences.
    - Specify argumentation module assumptions.
    - Validate argumentation module assumptions.
- Process Management
    - Check process compliance against standards.
    - Reuse assurance and certification assets across systems.
    - Reuse assurance and certification assets across standards and domains.

Knowledge Manager

Knowledge Manager [18] is a tool to manage all knowledge related to a system. It is based on the principle that knowledge is nowadays one of the most valuable assets in modern organizations, thus an appropriate management of the organization's knowledge is a key success factor. Knowledge must be gathered from many sources, stored into secure repositories, and accessed by the granted people at the right time. Knowledge further is a reusable asset in the systems and software projects.

Knowledge Manager stores knowledge into a well-designed repository referred to as System Knowledge Repository, which includes a System Knowledge Base as well as all assets. Knowledge Manager allows an easy access and maintenance, and its knowledge bases correspond to ontologies. Figure 5 shows the structure of an ontology in Knowledge Manager. An ontology consists of several layers, each depending on and extending the semantic information of the inner layer. The innermost layer (Terminology) corresponds to the terms of a domain together with their syntactic information. Relationships between the terms can be specified in the Conceptual model layer, as well as their semantics with clusters; e.g. the semantics of the terms 'car' and 'truck' can be 'system', and they specialise 'vehicle'. Patterns can then be developed to provide templates (aka boilerplates) for system information specification; the patterns refer to aspects of the two underlying layers. The Formalization layer includes information about how system information that matches a pattern will be semantically formalised and stored. Finally, at the Inference rules layer the data in all the other layers can be exploited for the specification of rules to derive new information, e.g. about the correctness of a system specification. More information about Knowledge Manager is available in [18].

Knowledge Manager supports assurance reuse in several ways:

- When assurance information is represented as an ontology in Knowledge Manager, the tool provides features for ontology elements reuse (e.g. for copy and for merge) and thus for assurance reuse.
- The data model for artefact representation that Knowledge Manager exploits is the basis for indexing any kind of system artefact and later use the tool's functionality for search.
- Not directly focused on reuse but related to it, assurance standards and compliance information can be represented with the semantic approach underlying Knowledge Manager. This information can be used in reuse scenarios.

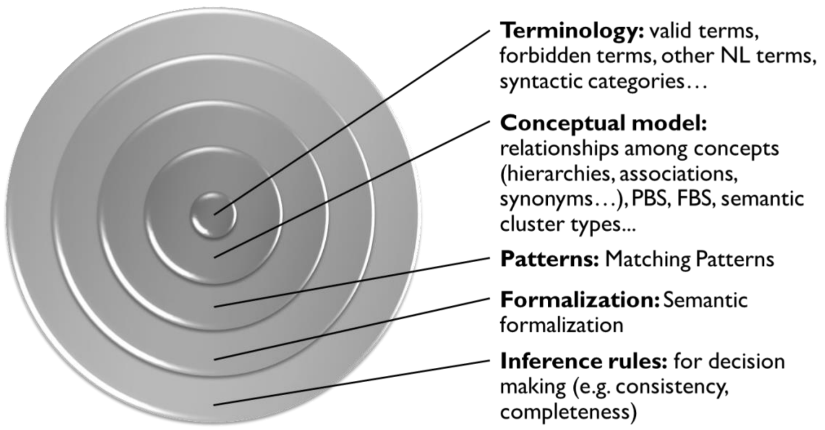

Terminology: valid terms, forbidden terms, other NL terms, syntactic categories…
Conceptual model: relationships among concepts (hierarchies, associations, synonyms…), PBS, FBS, semantic cluster types...
Patterns: Matching Patterns
Formalization: Semantic formalization
Inference rules: for decision making (e.g. consistency, completeness)

Figure 5. Ontology layers in Knowledge Manager

Safety Architect (*)

Safety Architect [61] is a risk analysis tool of complex systems using functional or physical architectures from usual modelling tools, such as Papyrus/CHESS tool. From a functional or physical definition of the system, Safety Architect allows performing a kind of local FMEA and automatically deducts the FTA corresponding to the identified feared events. As described in section 5.5 of D6.3 [12], Safety Architect supports the Diff/Merge methodology for the reuse of safety and security analysis artefacts. This Diff/Merge methodology can be applied during the re-import from AMASS Platform/CHESS tool to Safety Architect tool, as described in Section 3.10 of this document. Indeed, the synthesis view contains all the analysis artefacts differences between the source model and the target model, and the user can reuse differences to merge from the source model to the target model.

Elasticsearch & Kibana (*)

Elasticsearch [58] is "a distributed, JSON-based search and analytics engine designed for horizontal scalability, maximum reliability, and easy management". It is based on an open-source stack and basically can be used to "take data from any source, in any format, and search, analyse, and visualize it in real time" (as the Elastic site claims) [59]. The database is schema-free and supports the indexing and querying of arbitrary documents. It offers an HTTP based REST API and can be accessed by all major programming languages.

Within AMASS, Elasticsearch is used to index *any kind* of EMF based data supporting files but also CDO repositories. The main objective is to offer a free text search but also to build value added services and apps atop of it. There is a web-based general-purpose search app available and the Elasticsearch data are used by the Reuse Assistant. The Kibana software [60], basically a tool to visualize any kind of Elasticsearch data and to manage and navigate the Elasticsearch Stack visually, can be used to create custom dashboards and analytics.

Regorous (*)

Regorous Process Designer (for simplicity called only Regorous) [64] is a process compliance checker, which is part of the of the NICTA's Regorous Tool suite[1]. It assists analysts during the "*Process design phase of the process lifecycle*" with mapping regulations to specific process and process steps, so that processes can be designed or re-designed in a compliant way. Regorous is the result of the implementation of the approach for process compliance based on *the compliance-by-design methodology*, proposed in [65] and [66]. The compliance by design methodology enables the verification of compliance of a process with a set of rules, before the process is executed. To check whether a process is compliant with a relevant regulation, Regorous requires an annotated process model and the formal representation of the regulation (See Figure 6). The annotations are attached to the tasks of the process (Logical state representation), and they can be used to record the data, resources and other information. A process model is a self-contained, temporal and logical order in which a set of activities are expected to be executed to achieve a business goal.

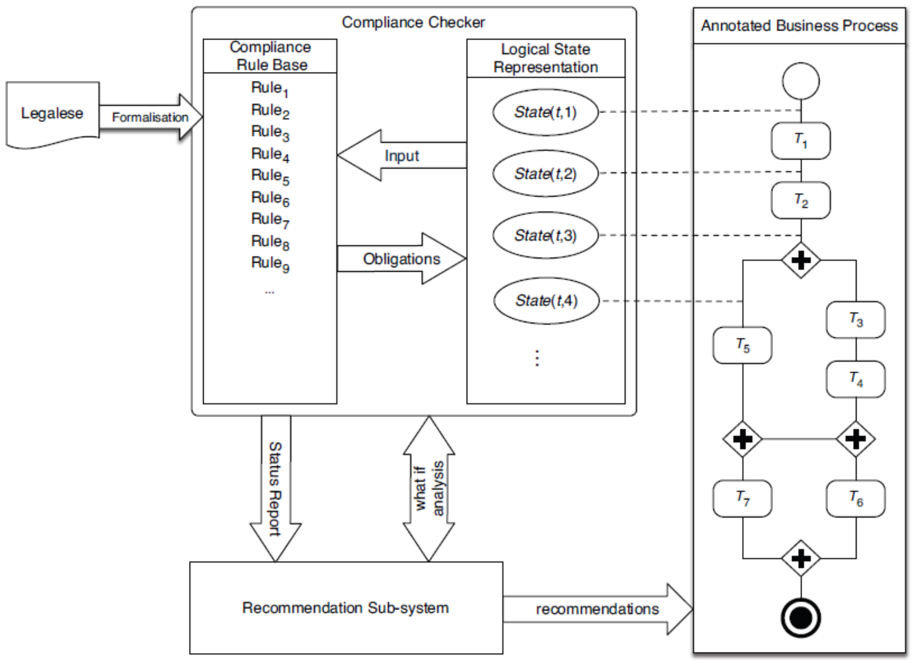


Figure 6. Regorous Abstract Framework

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

3. Methodological Guide

3.1 Workflow and Process Overview

The following Figure 7 shows a general workflow and an overview about related reuse methodologies elaborated in WP6.

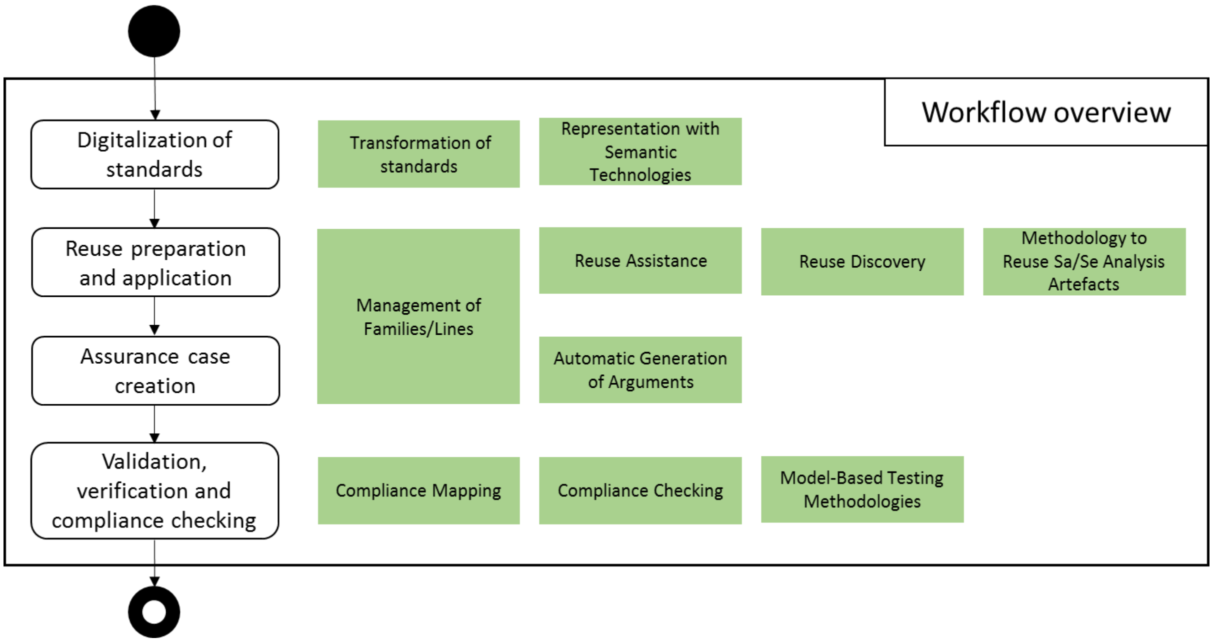

Workflow overview
Digitalization of standards
Reuse preparation and application
Assurance case creation
Validation, verification and compliance checking
Transformation of standards
Representation with Semantic Technologies
Management of Families/Lines
Reuse Assistance
Reuse Discovery
Methodology to Reuse Sa/Se Analysis Artefacts
Automatic Generation of Arguments
Compliance Mapping
Compliance Checking
Model-Based Testing Methodologies

Figure 7. Workflow and related reuse methodologies

3.1.1 Reuse assistance

Figure 8 shows the steps to follow a cross-system reuse using the OpenCert Reuse Assistant. The steps to work on this functionality are:

1. The user opens the newly created assurance project (target project), starts the Reuse Assistant, and selects the reusable assurance project (source project).
2. A number of assurance models from the source project are available for navigation, including evidence, process, argumentation and baseline (compliance information) models that can be individually selected.
3. Once a specific model is selected in the source project, the user invokes the reuse discovery function to search model elements according to specific criteria or selects model elements manually in a tree view with check boxes.
4. Once selected all the desired model elements to be reused from the source assurance project, the user can copy them to the target one executing the reuse operation.

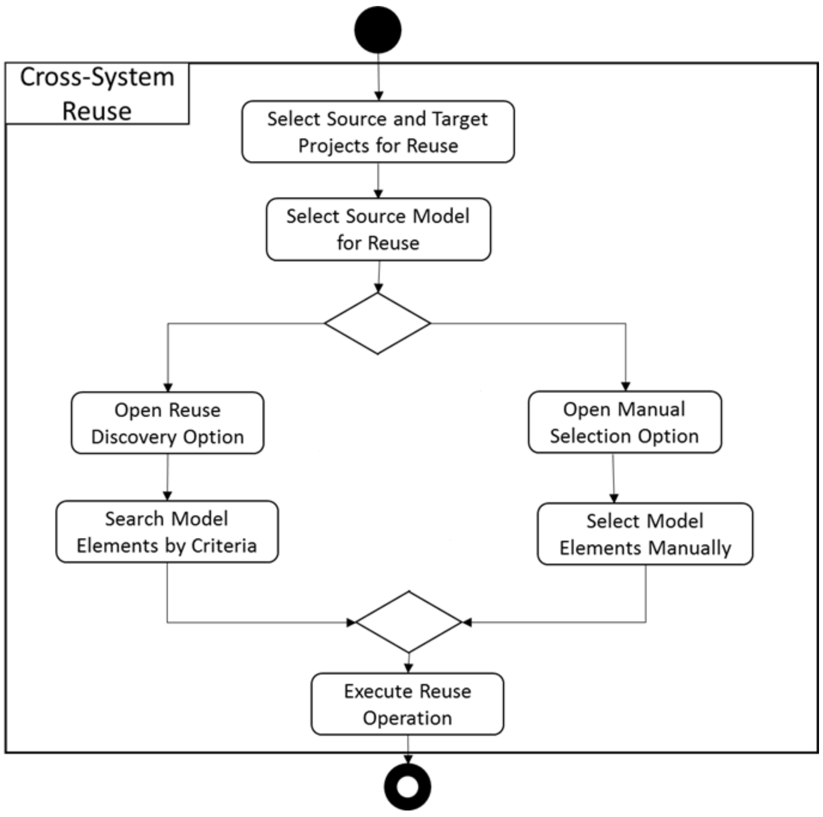

Cross-System Reuse
Select Source and Target Projects for Reuse
Select Source Model for Reuse
Open Reuse Discovery Option
Open Manual Selection Option
Search Model Elements by Criteria
Select Model Elements Manually
Execute Reuse Operation

Figure 8. Sub-activities related to the Cross-System Reuse using the Reuse Assistant

The workflow presented in Figure 8 is developed in section 3.2.1.

Figure 9 shows the steps to prepare the cross-standard reuse using the OpenCert Standards Editor. Three parallel activity flows describe the following aspects:

1. An expert on the source standard (i.e., the standard to which the source reuse project complies) must capture a Reference Framework model as edited by the OpenCert tools. Before starting to capture an interpretation of such standard, the expert would need to manually work on a conceptual model of this standard. The conceptual model must ensure that the different concepts from a standard are mapped to Reference Framework concepts as well as to the principles to demonstrate compliance.
2. An expert on the target standard (i.e., the standard to which the target reuse project complies) must capture a Reference Framework model as edited by the OpenCert tools. Before starting to capture an interpretation of such standard, the expert would need to manually work on a conceptual model of this standard. The conceptual models must ensure that the different concepts from a standard are mapped to Reference Framework concepts as well as to the principles to demonstrate compliance.
3. An expert on both standards must map concepts (activities, artefacts, and requirements) from source and target standards by using OpenCert Equivalence Maps. As a result of this activity, a model of Equivalence Maps between source and target Reference Frameworks will be generated.

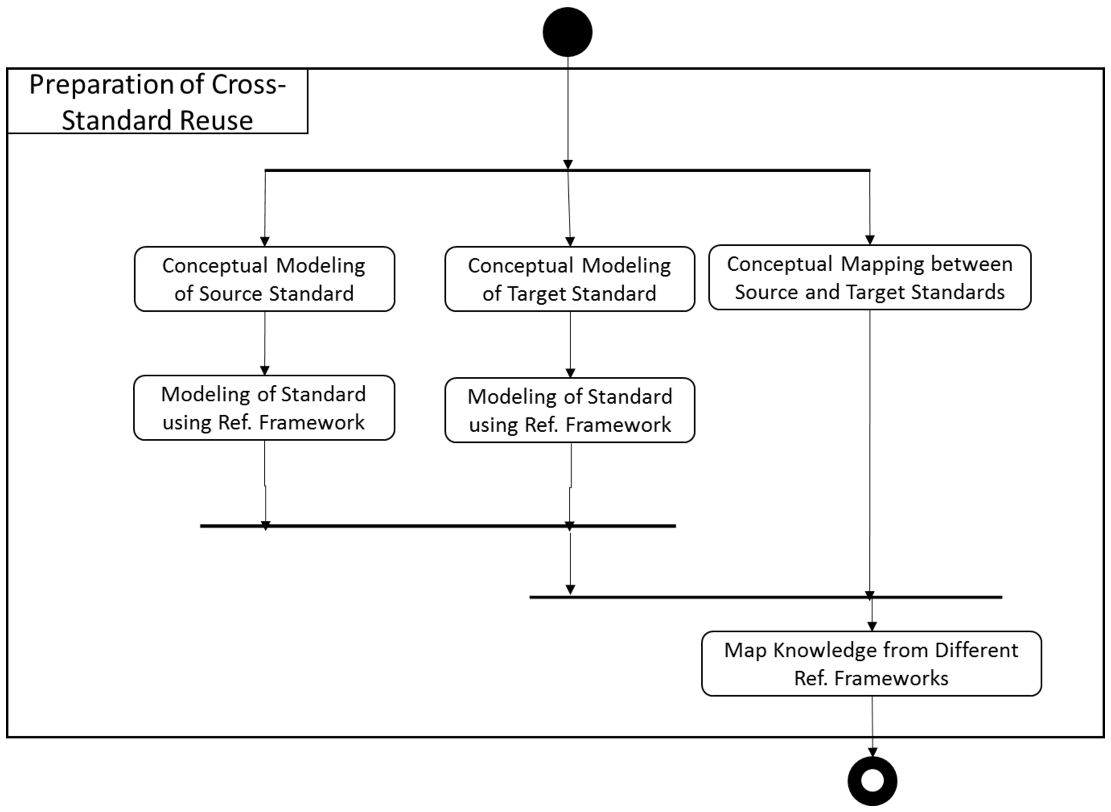

Preparation of Cross-Standard Reuse
Conceptual Modeling of Source Standard
Conceptual Modeling of Target Standard
Conceptual Mapping between Source and Target Standards
Modeling of Standard using Ref. Framework
Modeling of Standard using Ref. Framework
Map Knowledge from Different Ref. Frameworks

Figure 9. Sub-activities related to the Preparation of Cross-Standard Reuse

Figure 10 shows the steps for the actual cross-standard reuse. This prototype iteration is centred on evidence reuse. Two different flows apply related to the source and target projects for the reuse operation.

For the Source assurance project, the following steps must be followed:

1. Visualization of the evidence required to be provided, as prescribed by the source Reference Framework and definition of any additional evidence to be provided, not explicitly mentioned in the Reference Framework. This information is located in the Baseline Model(s) for the assurance project.
2. Create an Artefact Model and a Repository to store source project evidence documents in a centralized way.
3. Collect evidence documents into the Repository, during the assurance project life.
4. For each artefact generated, specify all its characteristics including owner, version, file location, etc. This also includes defining hierarchical dependencies between evidences and creating traceability links between evidences.
5. During the project life, some of these artefacts would need to be evaluated (e.g., reviews, validations). The AMASS Platform allow users to store evaluation results and the artefact status, including any event or action occurred around artefacts (changes of status, updates, etc.).

For the Target assurance project, the following steps must be followed:

1. Visualization of the evidence required to be provided, as prescribed by the target Reference Framework and definition of any additional evidence to be provided, not explicitly mentioned in the Reference Framework. This information is located in the Baseline Model(s) for the assurance project.
2. Create an Artefact Model and a Repository to store evidence documents in a centralized way.
3. Some evidential information could be potentially reused from the source assurance project, so in order to explore reuse opportunities, AMASS tools display to users the equivalence maps between

the source and target standards, as well as the actual artefacts that could be reused from the source assurance project.

4. The user will evaluate any implication of reusing artefacts, including post conditions or derived obligations to accomplish as result of reusing an artefact from the source assurance project. Any individual artefact and its associated information about compliance will be then reused from the source project into the target project.
5. Collect evidence documents into the repository, during the assurance project life.
6. For each generated artefact, specify all its characteristics including owner, version, file location, etc. This also includes defining hierarchical dependencies between evidences and creating traceability links between evidences.
7. During the project life, some of these artefacts would need to be evaluated (e.g., reviews, validations). The AMASS Platform allow users to store evaluation results and the artefact status, including any event or action occurred around artefacts (changes of status, updates, etc.).

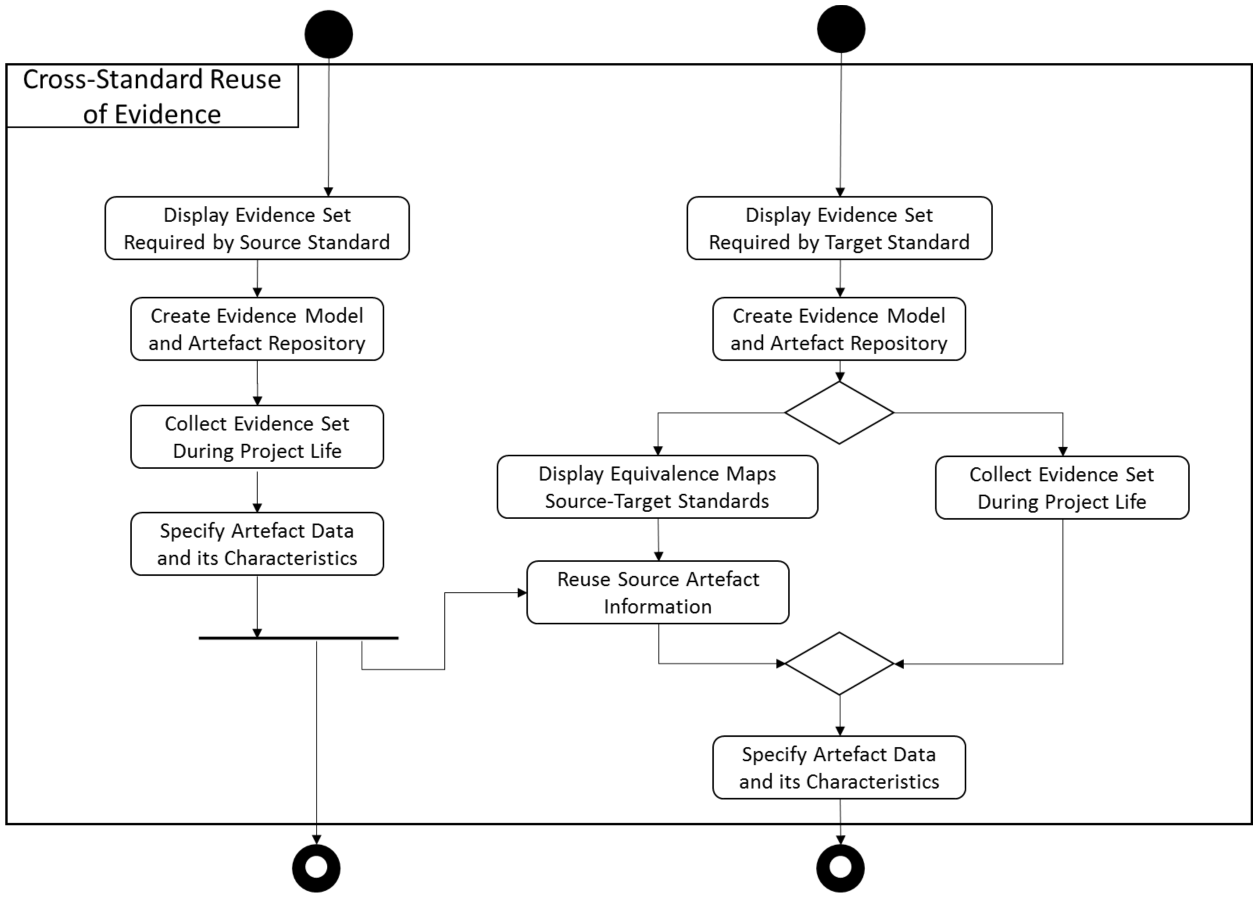

Cross-Standard Reuse of Evidence
Display Evidence Set Required by Source Standard
Create Evidence Model and Artefact Repository
Collect Evidence Set During Project Life
Specify Artefact Data and its Characteristics
Display Evidence Set Required by Target Standard
Create Evidence Model and Artefact Repository
Display Equivalence Maps Source-Target Standards
Collect Evidence Set During Project Life
Reuse Source Artefact Information
Specify Artefact Data and its Characteristics

Figure 10. Sub-activities related to the Cross-Standard Reuse of Evidence

The workflow presented in Figure 9 and Figure 10 is developed in section 3.2.2.

3.1.2 Management of families/lines

The workflow shown in Figure 11 illustrates the steps involved in the process variability management. At first, the path containing a method library is specified. After that, the method library is copied and the problems in XMI files are resolved. This is done for supporting the variability management with the BVR editors: VSpec, Resolution and Realization. To perform the resolution, engineers have to specify the desired true/false choices for the specific configuration. In order to derive the configuration, the execute option in particular resolution is selected. It might be noted that the realization fragments have been specified over the .xmi files. Finally, the configured process model is exported back to the method library.

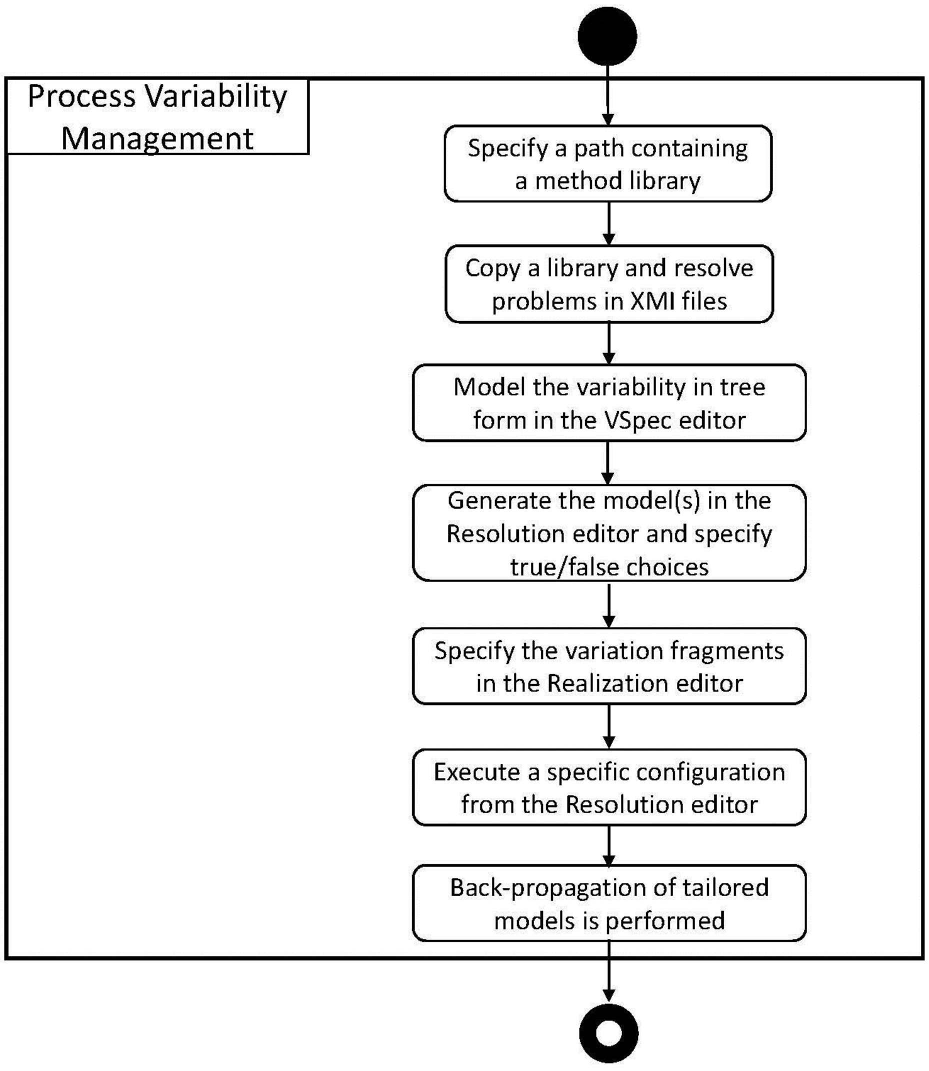

Process Variability Management
Specify a path containing a method library
Copy a library and resolve problems in XMI files
Model the variability in tree form in the VSpec editor
Generate the model(s) in the Resolution editor and specify true/false choices
Specify the variation fragments in the Realization editor
Execute a specific configuration from the Resolution editor
Back-propagation of tailored models is performed

Figure 11. Sub-activities related to the process variability management

The workflow presented in Figure 11 is developed in Section 3.3.1.

For the product and assurance case variability management, the models are either created or imported into the workspace. It might be noted that the variation points or fragments for product and assurance case are specified over the .uml and .arg files, respectively. The workflow presented in Figure 12 is developed in Sections 3.3.2 and 3.3.3.

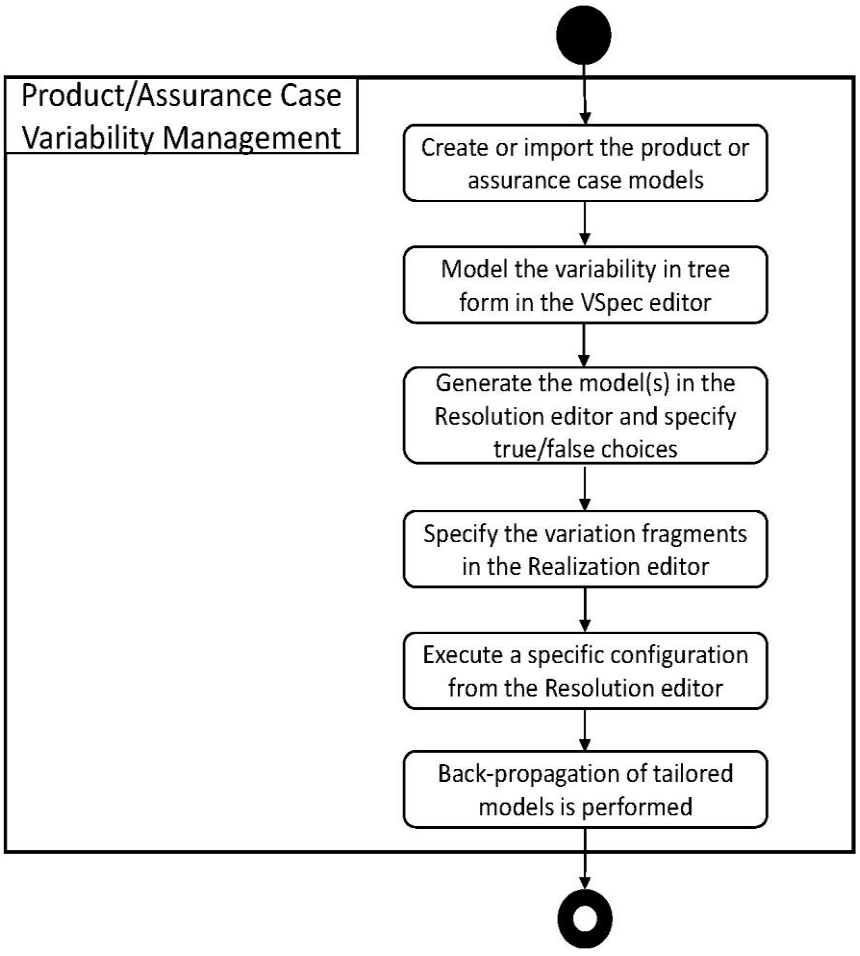

Product/Assurance Case Variability Management
Create or import the product or assurance case models
Model the variability in tree form in the VSpec editor
Generate the model(s) in the Resolution editor and specify true/false choices
Specify the variation fragments in the Realization editor
Execute a specific configuration from the Resolution editor
Back-propagation of tailored models is performed

Figure 12. Sub-activities related to the product/assurance case variability management

The first block of the Framework shown in Figure 13 contains identification of regulations and standards, and the second process development with EPF-C and BVR. The identification step identifies mandatory standards and additional relevant regulations. These are customer requirements and additional company specific regulations. The following activity is process development in EPF-C with the intention to define a process, which is ready for reuse and instantiation.

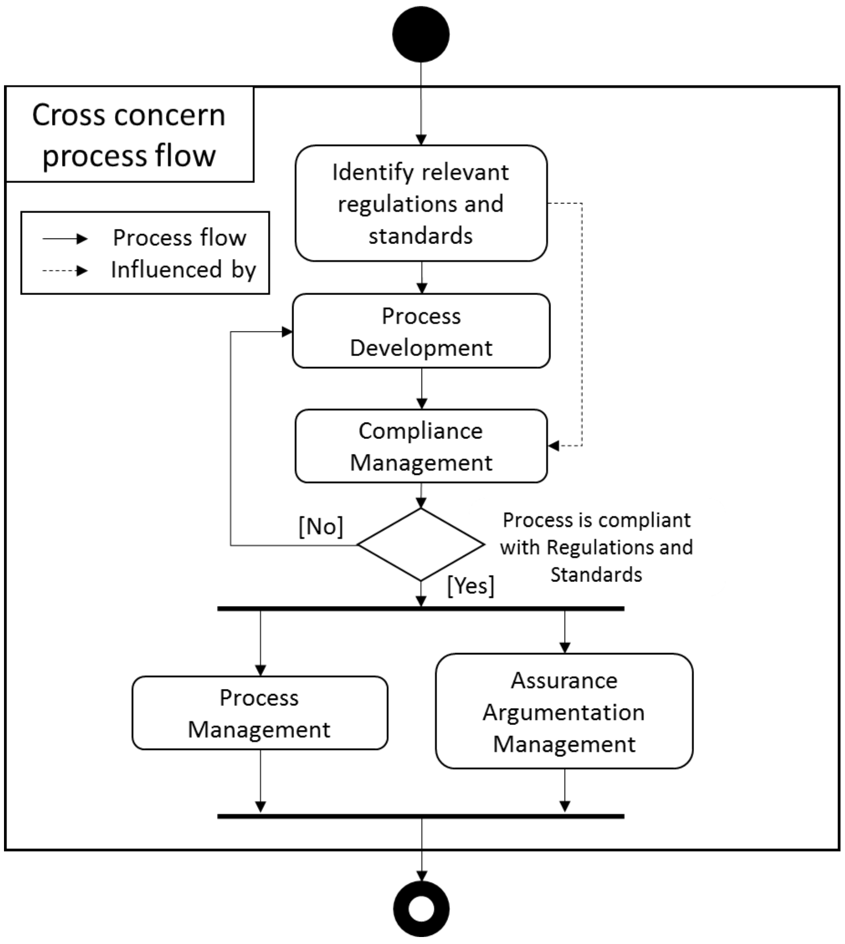

Cross concern process flow
Process flow
Influenced by
Identify relevant regulations and standards
Process Development
Compliance Management
[No]
Process is compliant with Regulations and Standards
[Yes]
Process Management
Assurance Argumentation Management

Figure 13. Big picture of process framework related to Cross concern reuse

The workflow in Figure 14 shows a way to develop reusable processes. As initial step, relevant standards are identified or, alternatively (drawn with dotted line), an existing engineering process is selected. Based on standards or on an existing process, the reusable generic base process is developed. It includes all activities, which might be necessary and all available methods. With the instantiation to a project specific process, a reduction takes place. All activities that are not supposed to be part of this specific process are removed. A check is necessary to be sure that the assumptions for the reused activities are valid for the new project. If the validation is passed, the project specific process, which reuses parts of the generic base process, is ready for execution. During process execution, engineers must deal with variabilities mainly caused by product specific decisions.

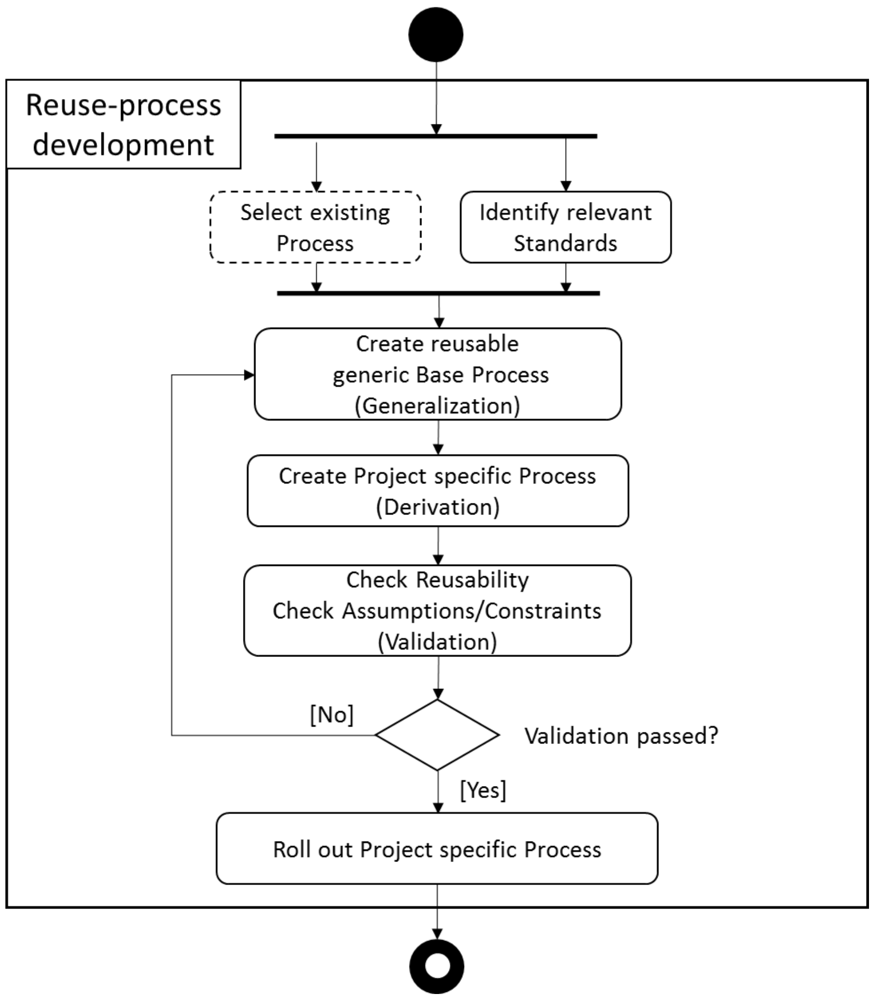

Reuse-process development
Select existing Process
Identify relevant Standards
Create reusable generic Base Process (Generalization)
Create Project specific Process (Derivation)
Check Reusability
Check Assumptions/Constraints
(Validation)
[No]
Validation passed?
[Yes]
Roll out Project specific Process

Figure 14. Sub-activities related to reuse-process development

The workflow presented in Figure 13 and Figure 14 is developed in section 3.3.1.1.

3.1.3 Automatic generation of arguments

The workflow shown in Figure 15 illustrates the steps related to the generation of a process-based argumentation model and diagram. For that, first the requirements and process model are created according to the standards. Then, the validation is performed to detect (fallacy) whether the process model contains the sufficient information corresponding to the key evidence for supporting the specific requirement. In case of omission of crucial information (i.e., insufficient evidence), recommendations are provided to aid users in resolving deviations. Based on the recommendation(s), users are expected to modify the process model. Finally, the modified process model is proceeded to generate the process-based argumentations. For this, the target assurance project is selected to store the argumentations.

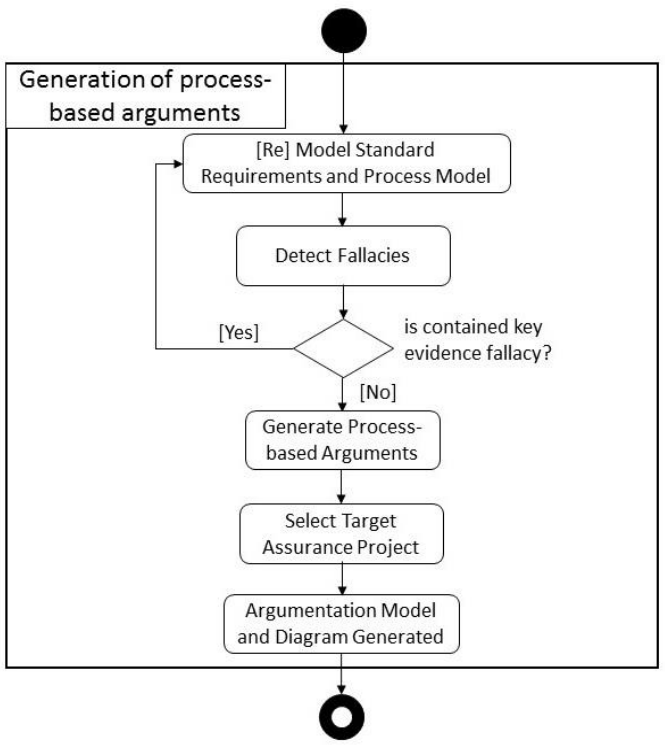

Generation of process-based arguments
[Re] Model Standard Requirements and Process Model
Detect Fallacies
[Yes]
is contained key evidence fallacy?
[No]
Generate Process-based Arguments
Select Target Assurance Project
Argumentation Model and Diagram Generated

Figure 15. Sub-activities related to the generation of process-based arguments

The workflow presented in Figure 15 is developed in Section 3.4.1.

3.1.4 Transformation of Standard's Requirement

The workflow shown in Figure 16 illustrates the steps related to the transformation of standard's requirements to a Baseline model and diagram. In this context, first the requirements and process model are created according to the standards. Then, the modelled requirements are proceeded to generate the baselines. For this, a target assurance project is selected to store the Baseline model and diagram.

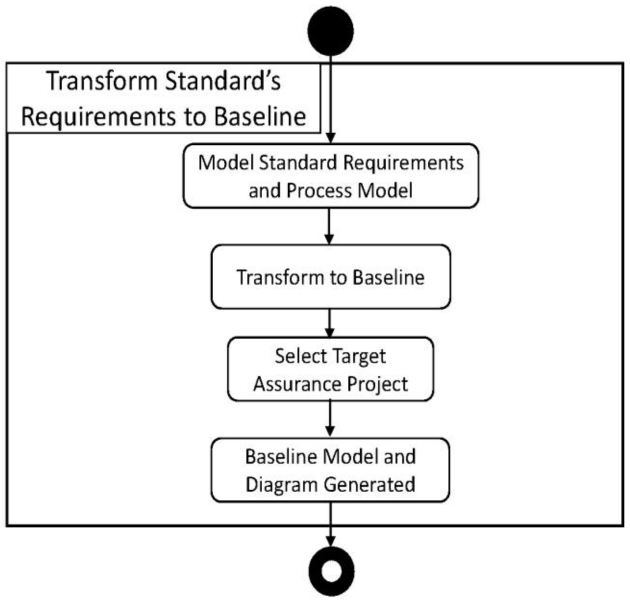

Transform Standard's Requirements to Baseline
Model Standard Requirements and Process Model
Transform to Baseline
Select Target Assurance Project
Baseline Model and Diagram Generated

Figure 16. Sub-activities related to the tranformation of standard's requirements to Baseline model

The methodological guide for the transformation of standard's requirements to Baseline is presented in Section 3.5.

3.1.5 Compliance Checking Workflow (*)

The workflow shown in Figure 17 illustrates the steps related to automated compliance checking of process against safety standards.

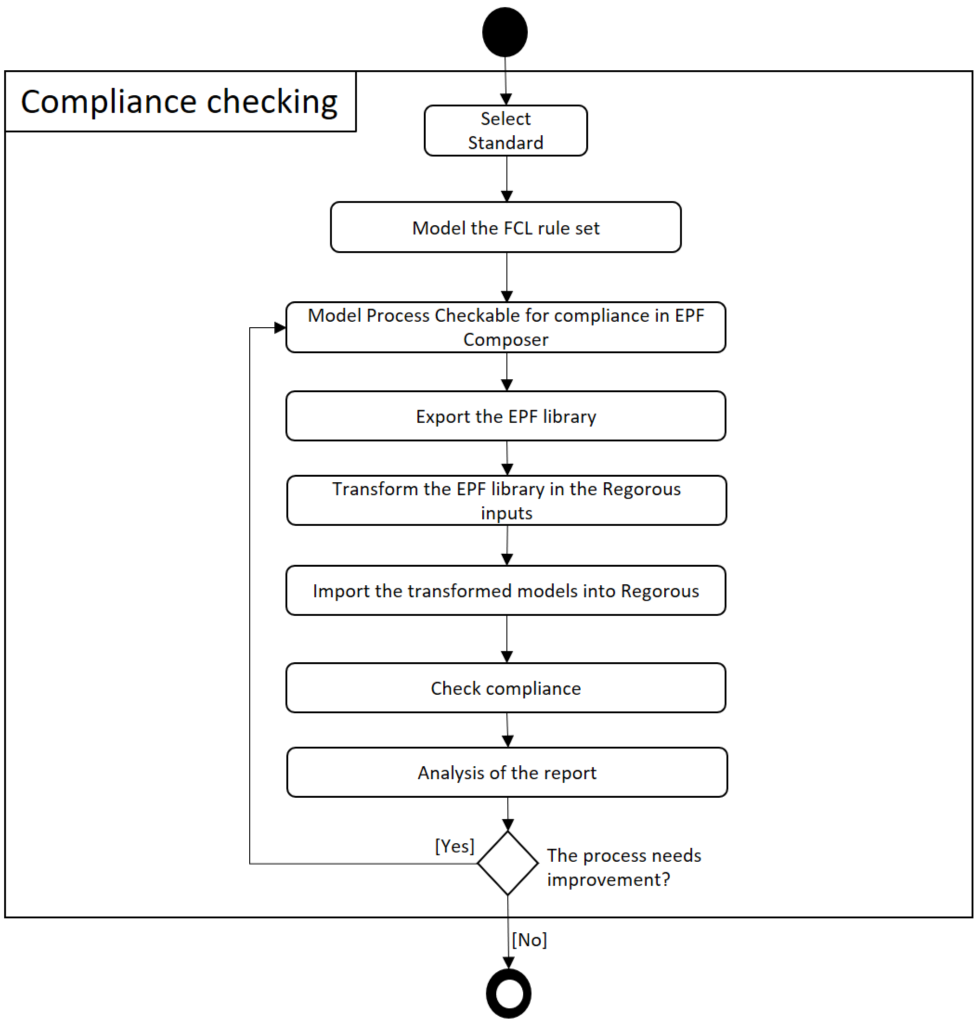

Compliance checking
Select Standard
Model the FCL rule set
Model Process Checkable for compliance in EPF Composer
Export the EPF library
Transform the EPF library in the Regorous inputs
Import the transformed models into Regorous
Check compliance
Analysis of the report
[Yes]
The process needs improvement?
[No]

Figure 17. Workflow for compliance checking modelling

The methodological guide related to compliance checking is presented in Section 3.7.

3.1.6 Representation of safety standards with semantic technologies

Figure 18 shows the workflow for representation of safety standards with semantic technologies. The workflow is developed in section 3.9.

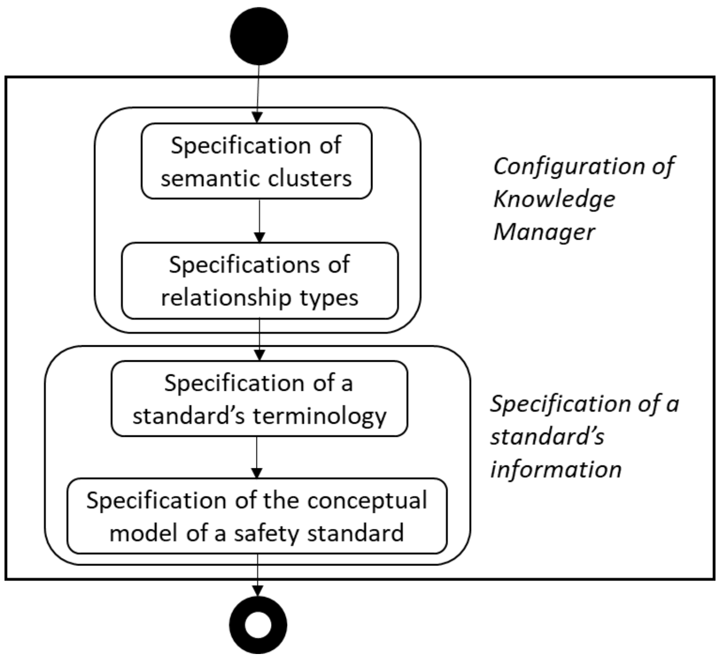

Specification of semantic clusters
Specifications of relationship types
Configuration of Knowledge Manager
Specification of a standard's terminology
Specification of the conceptual model of a safety standard
Specification of a standard's information

Figure 18. Workflow for the representation of safety standards with semantic technologies

3.2 Reuse Assistance (*)

The reuse assistance functionality concerns intra and cross-standard reuse of assurance and certification assets. The design concept for this functionality is described in D6.3 [12]. The AMASS Platform focuses on the reuse of the following process-based assurance assets:

- ***Compliance checks:*** any information related to the accomplishment of industry standards (i.e., information of level of compliance, compliance justification, traceability to claims or evidence).
- ***Artefacts:*** characterization of evidential artefacts (e.g., evidence attributes, evaluations, versions, and link to concrete artefact resources).
- ***Argumentations:*** complete argumentations or argumentation fragments**.**
- ***Activities:*** any information of activities executed as part of a reusable assurance project.

Product-based assets such as requirements, design artefacts, code, or even assurance case-based assets such as argumentation fragments have been excluded.

The concrete scenarios of reuse include (see Figure 19):

- ***Cross-Project reuse (intra standard or intra domain product upgrade):*** 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 on the basis of an existing one. Such a new system can include, for instance, some new components. It was assumed that reusable assurance assets are compliant with the same standards that are targeted in the new scenario.
- ***Cross-standard reuse (including cross-concern and cross-domain):*** 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). It also includes the case when the new assurance project must comply with a new standard in the same domain, a new version of an existing standard, or a different interpretation of a standard (e.g., by a different certification authority).

In the AMASS Platform, the Reuse Assistant tool does not cover reuse scenarios such as COTS or SEooC-like reuse.

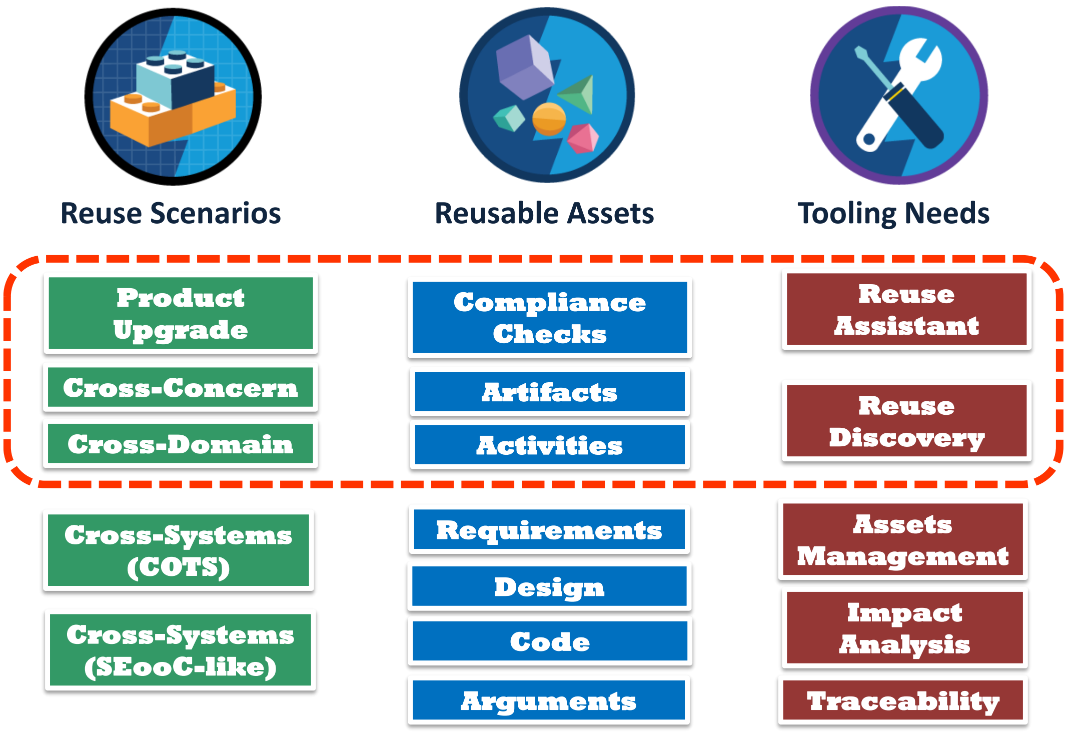


Figure 19. Reuse Assistant: Scope of Reuse in Assurance and Certification

AMASS will support users to understand whether reuse of the assurance assets is reasonable or determine what further assurance activities (engineering, V&V, or compliance activities) are required to justify compliance in the new scenario.

3.2.1 Cross-Project (intra standard or intra domain product upgrade)

This functionality is accessed by means of 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. The AMASS Platform also allows users, through a context menu (see Figure 20), to search a subset of evidence model elements according some search criteria introduced using one of the two existing reuse discovery engines, avoiding or validating with this the manual selection of reusable assets. The first engine is related to the OSLC-KM tool [19] and the second one uses Elastic Search [59].

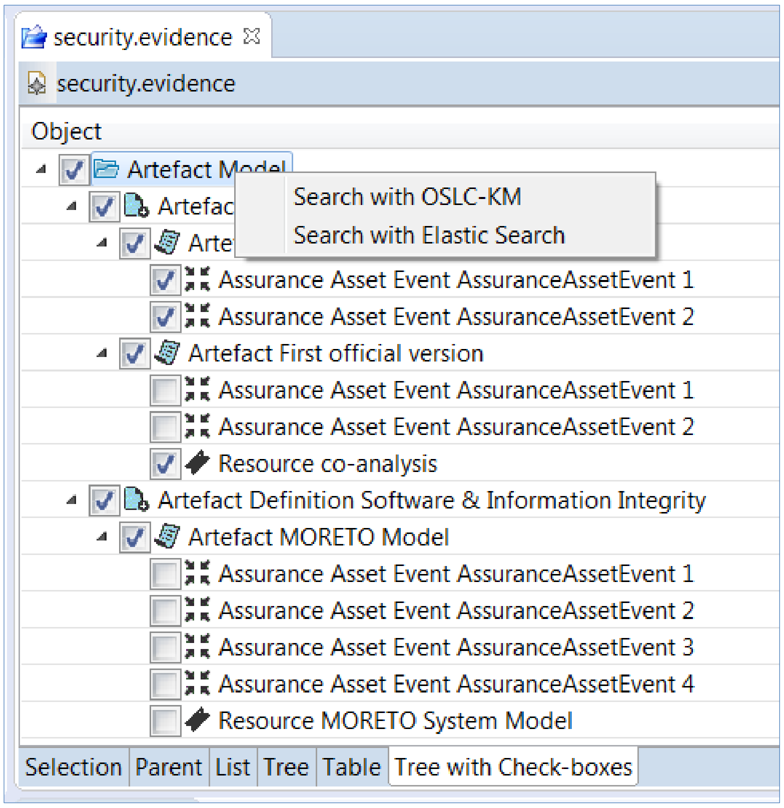
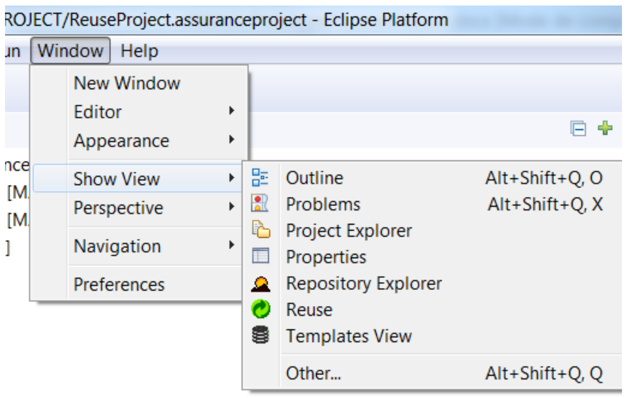

security.evidence
security.evidence
Object
Artefact Model
Artefac
Arte
Search with OSLC-KM
Search with Elastic Search
Assurance Asset Event AssuranceAssetEvent 1
Assurance Asset Event AssuranceAssetEvent 2
Artefact First official version
Assurance Asset Event AssuranceAssetEvent 1
Assurance Asset Event AssuranceAssetEvent 2
Resource co-analysis
Artefact Definition Software & Information Integrity
Artefact MORETO Model
Assurance Asset Event AssuranceAssetEvent 1
Assurance Asset Event AssuranceAssetEvent 2
Assurance Asset Event AssuranceAssetEvent 3
Assurance Asset Event AssuranceAssetEvent 4
Resource MORETO System Model
Selection
Parent
List
Tree
Table
Tree with Check-boxes

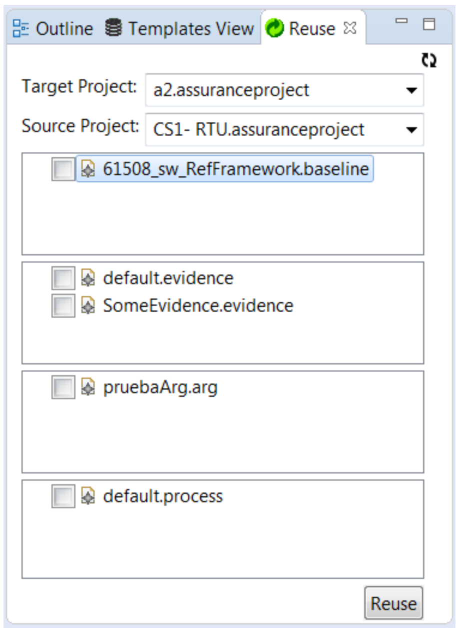
Figure 20. Context menu to choose an assets discovery engine

Figure 21 shows how to open the Reuse View and Figure 22 shows 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.

The steps to work on this functionality are:

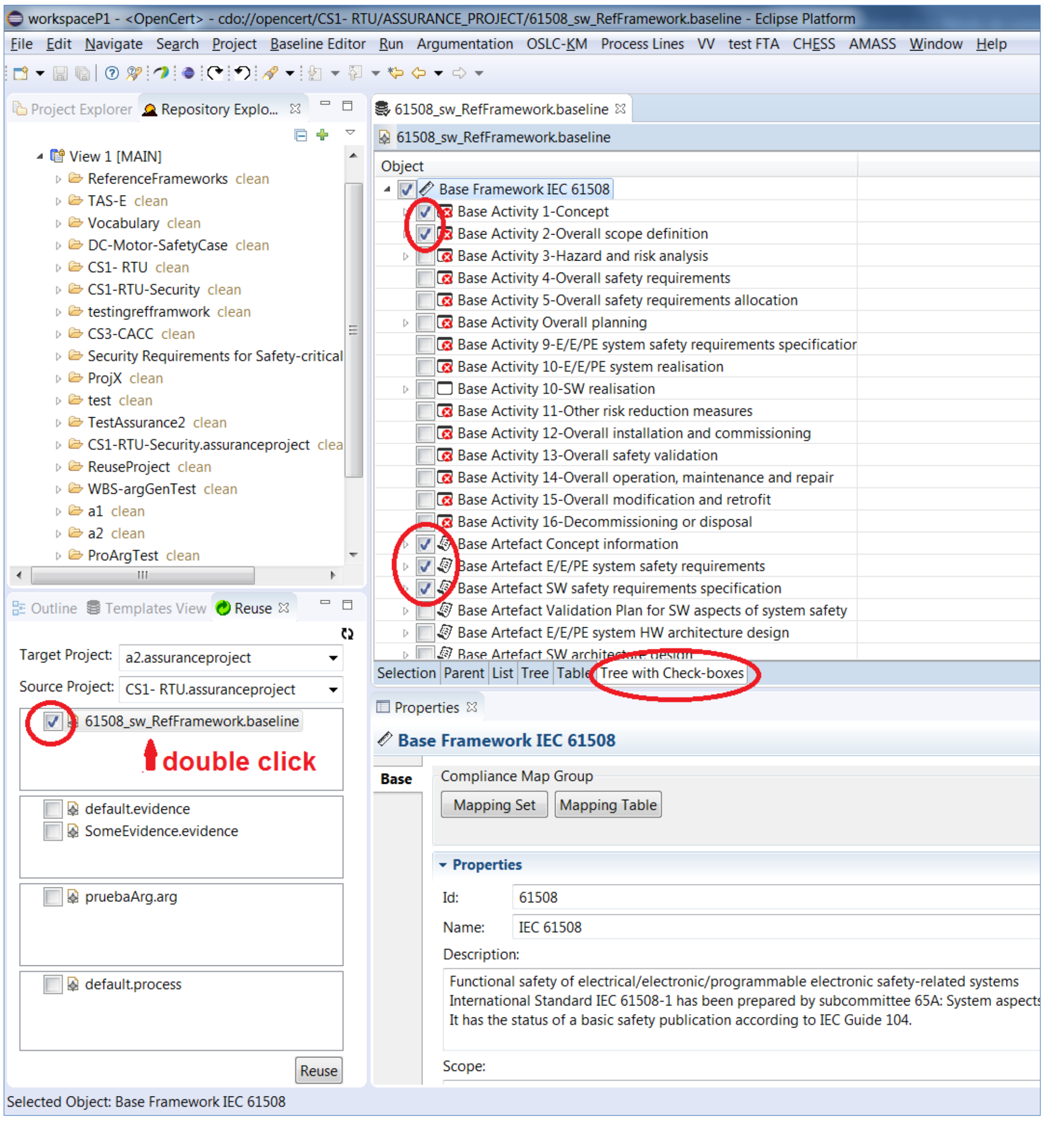
1. The user opens the reuse assistant view.
2. The user selects the assurance project that will receive the elements to reuse (target project), normally newly created assurance project.
3. The user and selects the reusable assurance project (source project).
4. A number of assurance models from the source project are available for navigation, including evidence, process, argumentation and baseline (compliance information) models and can be individually selected.
5. If the user checks the model, it will be reused completely.
6. Double clicking over a model of the lists, will open an editor to allow the user to select manually model parts (see Figure 22).




Target Project: a2.assuranceproject

Source Project: CS1- RTU.assuranceproject

Figure 21. Cross-project Reuse View


workspaceP1 - <OpenCert> - cdo://opencert/CS1- RTU/ASSURANCE_PROJECT/61508_sw_RefFramework.baseline - Eclipse Platform

File Edit Navigate Search Project Baseline Editor Run Argumentation OSLC-KM Process Lines VV test FTA CHESS AMASS Window Help

Project Explorer | Repository Explo...

View 1 [MAIN]
- ReferenceFrameworks clean
- TAS-E clean
- Vocabulary clean
- DC-Motor-SafetyCase clean
- CS1- RTU clean
- CS1-RTU-Security clean
- testingrefframwork clean
- CS3-CACC clean
- Security Requirements for Safety-critical
- ProjX clean
- test clean
- TestAssurance2 clean
- CS1-RTU-Security.assuranceproject clea
- ReuseProject clean
- WBS-argGenTest clean
- a1 clean
- a2 clean
- ProArgTest clean

Outline | Templates View | Reuse

Target Project: a2.assuranceproject

Source Project: CS1- RTU.assuranceproject

- [✓] 61508_sw_RefFramework.baseline

double click

- [ ] default.evidence
- [ ] SomeEvidence.evidence

- [ ] pruebaArg.arg

- [ ] default.process

Reuse

61508_sw_RefFramework.baseline

Object
- [✓] Base Framework IEC 61508
  - [✓] Base Activity 1-Concept
  - [✓] Base Activity 2-Overall scope definition
  - [ ] Base Activity 3-Hazard and risk analysis
  - [ ] Base Activity 4-Overall safety requirements
  - [ ] Base Activity 5-Overall safety requirements allocation
  - [ ] Base Activity Overall planning
  - [ ] Base Activity 9-E/E/PE system safety requirements specificatior
  - [ ] Base Activity 10-E/E/PE system realisation
  - [ ] Base Activity 10-SW realisation
  - [ ] Base Activity 11-Other risk reduction measures
  - [ ] Base Activity 12-Overall installation and commissioning
  - [ ] Base Activity 13-Overall safety validation
  - [ ] Base Activity 14-Overall operation, maintenance and repair
  - [ ] Base Activity 15-Overall modification and retrofit
  - [ ] Base Activity 16-Decommissioning or disposal
  - [✓] Base Artefact Concept information
  - [✓] Base Artefact E/E/PE system safety requirements
  - [✓] Base Artefact SW safety requirements specification
  - [ ] Base Artefact Validation Plan for SW aspects of system safety
  - [ ] Base Artefact E/E/PE system HW architecture design
  - [ ] Base Artefact SW architecture design

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

Properties

Base Framework IEC 61508

Base

Compliance Map Group

Mapping Set | Mapping Table

Properties

Id: 61508

Name: IEC 61508

Description:

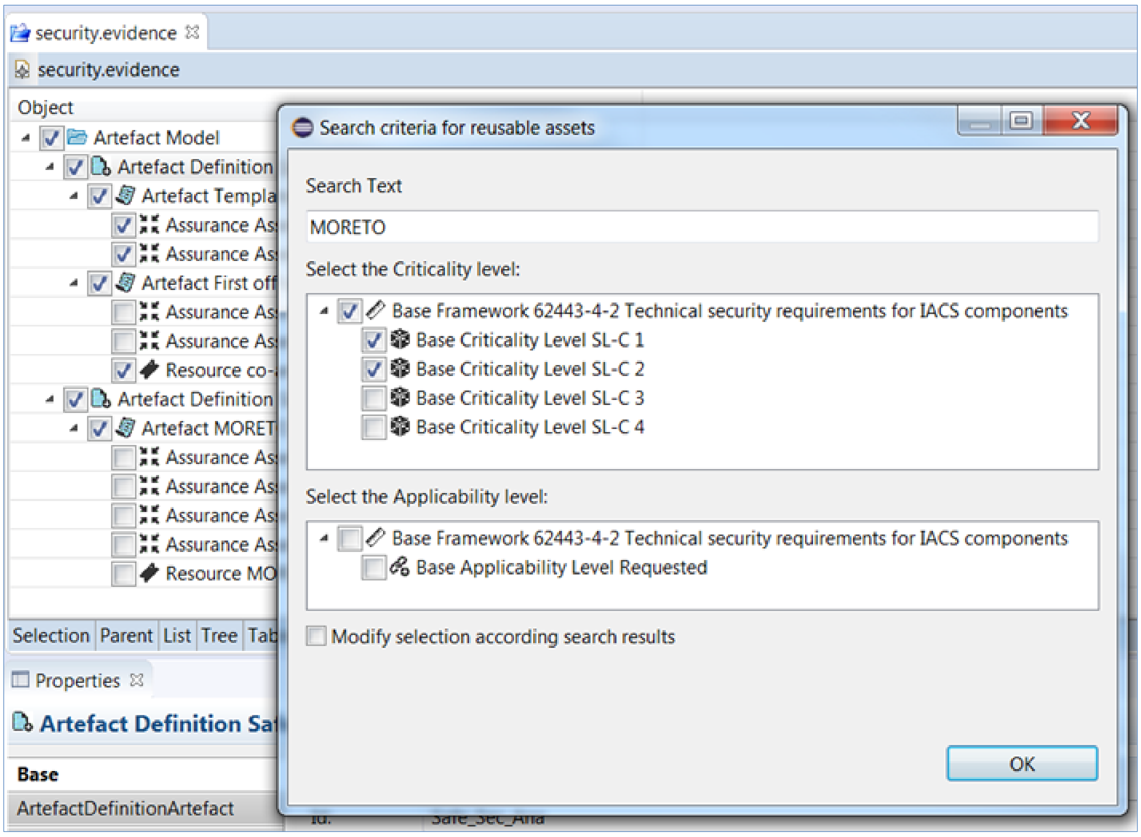
Functional safety of electrical/electronic/programmable electronic safety-related systems
International Standard IEC 61508-1 has been prepared by subcommittee 65A: System aspects
It has the status of a basic safety publication according to IEC Guide 104.

Scope:

Selected Object: Base Framework IEC 61508

Figure 22. Interaction with the Reuse View

7. The user can search reusable assets in the selected evidence model according to a text and/or a selected critically level and an introduced text (the applicability level is not used) (see Figure 23). The Elastic Search Engine will only use the introduced text as search criteria and the OSLC-KM both, the introduced text and the selected criticality levels specified by the standard that the source project is compliant with. The user has an option to select automatically evidence model elements according to the results of the search and modifying the previous subset of evidence model elements selected.


security.evidence

Search criteria for reusable assets

Search Text

MORETO

Select the Criticality level:

- [✓] Base Framework 62443-4-2 Technical security requirements for IACS components
  - [✓] Base Criticality Level SL-C 1
  - [✓] Base Criticality Level SL-C 2
  - [ ] Base Criticality Level SL-C 3
  - [ ] Base Criticality Level SL-C 4

Select the Applicability level:

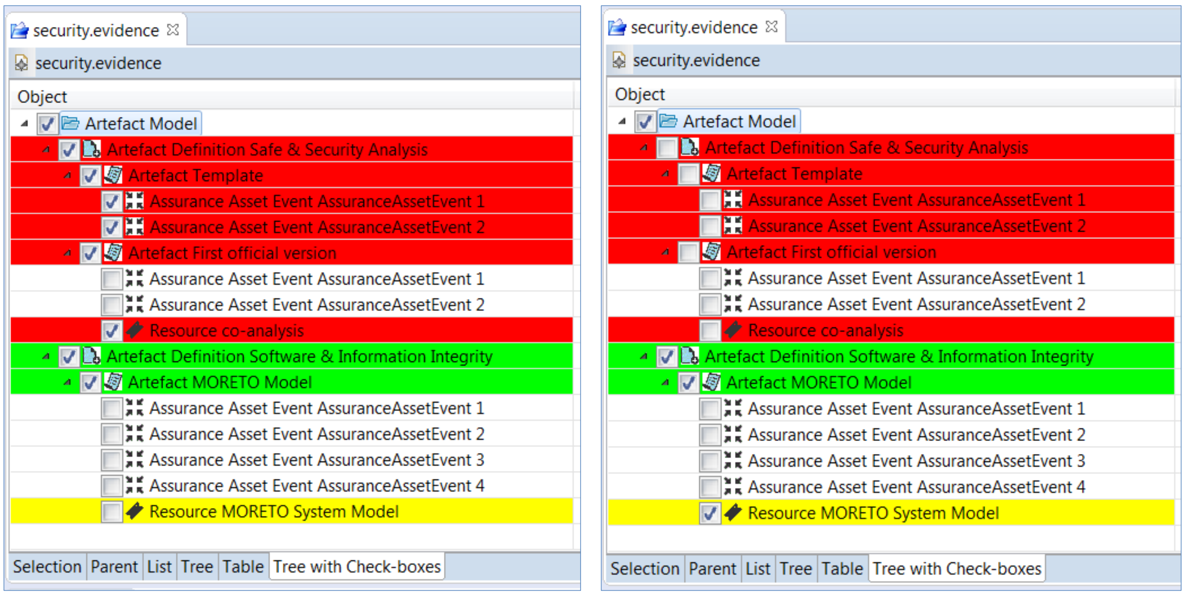
- [ ] Base Framework 62443-4-2 Technical security requirements for IACS components
  - [ ] Base Applicability Level Requested

[ ] Modify selection according search results

Figure 23. Window to introduce the search parameters

8. The results are shown to the user with a colour code (see Figure 24).

    - In green. The selected evidence model element, before executing the search, is a good candidate for reusing according to the introduced searching criteria.
    - In red. The selected evidence model element, before executing the search, does not comply with the introduced searching criteria and therefore should be unselected to avoid its reuse.
    - In yellow. The unselected evidence model element, before executing the search, is a good candidate for reusing and therefore should be selected to be reused.



security.evidence

Object

- ✓ Artefact Model
  - ✓ Artefact Definition Safe & Security Analysis
    - ✓ Artefact Template
      - ✓ Assurance Asset Event AssuranceAssetEvent 1
      - ✓ Assurance Asset Event AssuranceAssetEvent 2
    - ✓ Artefact First official version
      - ☐ Assurance Asset Event AssuranceAssetEvent 1
      - ☐ Assurance Asset Event AssuranceAssetEvent 2
      - ✓ Resource co-analysis
  - ✓ Artefact Definition Software & Information Integrity
    - ✓ Artefact MORETO Model
      - ☐ Assurance Asset Event AssuranceAssetEvent 1
      - ☐ Assurance Asset Event AssuranceAssetEvent 2
      - ☐ Assurance Asset Event AssuranceAssetEvent 3
      - ☐ Assurance Asset Event AssuranceAssetEvent 4
      - ☐ Resource MORETO System Model

security.evidence

Object

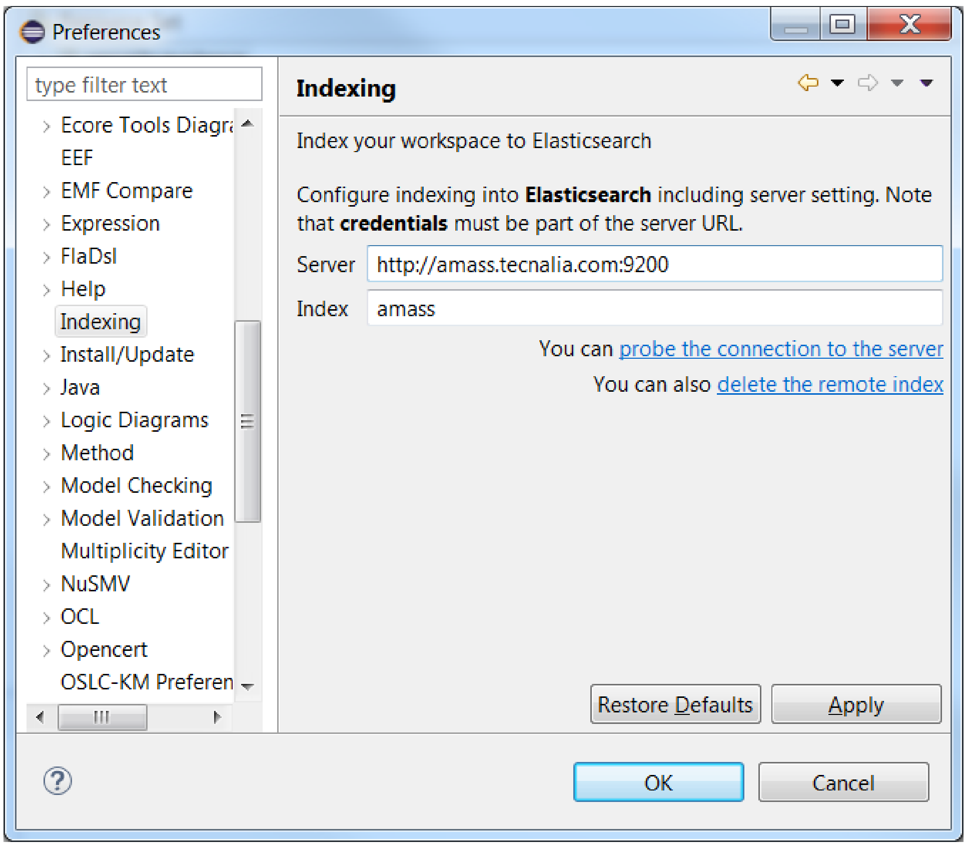
- ✓ Artefact Model
  - ☐ Artefact Definition Safe & Security Analysis
    - ☐ Artefact Template
      - ☐ Assurance Asset Event AssuranceAssetEvent 1
      - ☐ Assurance Asset Event AssuranceAssetEvent 2
    - ☐ Artefact First official version
      - ☐ Assurance Asset Event AssuranceAssetEvent 1
      - ☐ Assurance Asset Event AssuranceAssetEvent 2
      - ☐ Resource co-analysis
  - ✓ Artefact Definition Software & Information Integrity
    - ✓ Artefact MORETO Model
      - ☐ Assurance Asset Event AssuranceAssetEvent 1
      - ☐ Assurance Asset Event AssuranceAssetEvent 2
      - ☐ Assurance Asset Event AssuranceAssetEvent 3
      - ☐ Assurance Asset Event AssuranceAssetEvent 4
      - ✓ Resource MORETO System Model

Figure 24. Results of a search (the right one with automatic selection option active)

9. Once selected all the desired model elements to be reused from the various models, the user must press the “Reuse” button to start the reuse operation.

3.2.1.1 Usage of Tools

Before using Elasticsearch, it’s necessary to configure the connection to the indexing server. The server may run locally or remotely. To configure this feature, go to Eclipse Preference Page and configure the server URL and the Index name, as shown in Figure 25.



Indexing

Index your workspace to Elasticsearch

Configure indexing into **Elasticsearch** including server setting. Note that **credentials** must be part of the server URL.

Server: http://amass.tecnalia.com:9200

Index: amass

You can probe the connection to the server

You can also delete the remote index

Figure 25. Configuration of the Elasticsearch indexing server

In the AMASS Platform, it is still necessary to index objects and models manually (e.g. the source evidence model) before executing the search over its information using the context menu (see Figure 26).

The user can index arbitrary artefacts (system models but also insurance case related metadata or safety case data). The indexer does not use any separate meta-meta language or schema to index the data, but instead completely relies on the metamodels of the artefacts directly. By doing that, the indexer benefits from the advantages of a schema free database as Elasticsearch. In future scenarios, it can be assumed that the data are automatically indexed once they are changed on the file system or they are committed to SVN or CDO database.

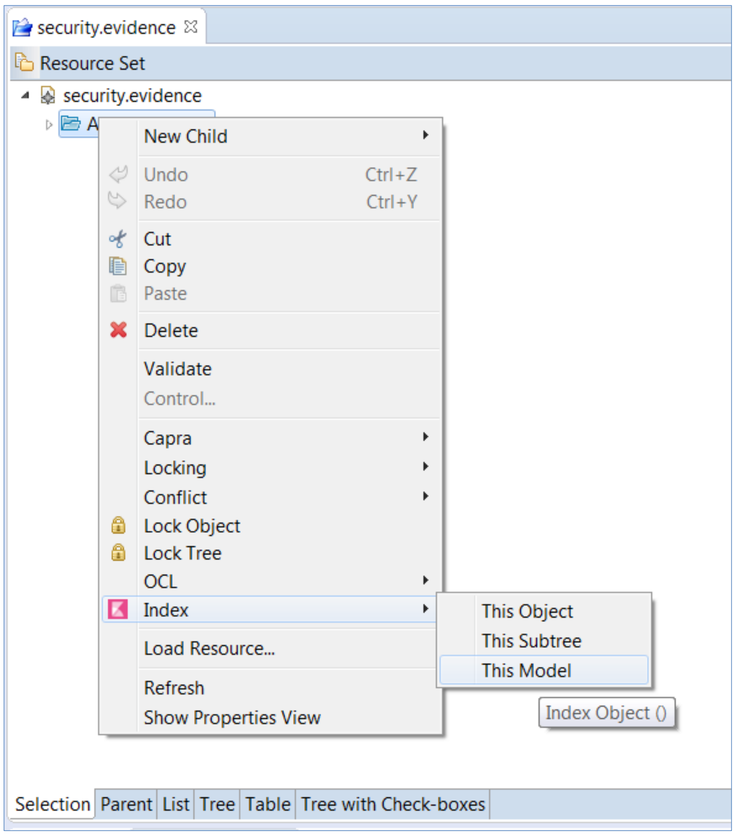


Figure 26. Index a model using Elasticsearch

3.2.2 Cross-Standard reuse (cross-concern or cross-domain)

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 tool provides information on the reuse opportunities as result of the equivalence relationships. Once the user 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.

According to the AMASS analysis, there are four main principles that should be taken into consideration when reusing artefacts across standards:

- the intent of a certification artefact must be taken into account when aiming at its reuse,
- maps must be established between the source standard (*reuse from*) and the target one (*reuse to*),
- project compliance must be determined (by means of maps), and
- needs and gaps resulting from certification artefact reuse must be determined.

These four principles are orderly described below.

1) Certification artefact intent

Certification artefact reuse is not a challenge per se. In theory, any artefact is reusable. The main reuse need stems from the fact that each standard has its own requirements to fulfil, and such requirements can vary among standards. For example, DO-178C [22] lists objectives (requirements) for the different software development processes, and some of its objectives are not fully addressed in EN 50128 [21]. When reusing a certification artefact produced in compliance with a source standard, it must be determined which requirements of the target standard are fulfilled. Certification artefact reuse can be regarded as the process targeted at determining which requirements of a given (target) standard are fulfilled when compliance with another (source) standard has been achieved.

The above need can only be met if the standard requirements, whose fulfilment a certification artefact contributes to, are recorded. These requirements correspond to the artefact intent: which properties are assured in the artefact, and thus why the artefact is necessary. For example, the DO-178C Software Requirements Data must include the performance criteria, timing requirements and constraints, and memory size constraints so that the artefact fulfils its intent (i.e., to show that such characteristics have been considered and specified).

The certification artefact intent is also based on the activities that use or produce the artefact. In general, and considering that activities use and produce several certification artefacts, the overall aim of an activity corresponds to a higher-level intent than the individual intent of the output artefacts of the activity. The achievement of this higher-level intent is also enabled by the individual intent of the input artefacts of the activity. For example, EN 50128 Integration Process (activity) aims at demonstrating that software and hardware interact correctly to perform their intended functions. To this end, the activity uses the Software Integration Test Specification and the Software/Hardware Integration Test Specification as input and produces the Software Integration Test Report and the Software/Hardware Integration Test Report as output.

2) Equivalence mapping between standards

In addition to recording the intent of the certification artefacts, it is also necessary to determine the equivalence between standards for certification artefact reuse. This can be done by means of maps that indicate the extent to which the requirements of the standards (e.g., requirement concerning the provision of a certain artefact) correspond (e.g., between EN 50128 Software Requirements Specification and DO-178C Software Requirements Data). Based on these mappings, the similarity and differences of the standards can be assessed, and thus how compliant a certification artefact is with respect to a given (target) standard according to its compliance with another (source) standard.

Three general types of maps can exist between the elements of two standards:

- ***Full map***: the elements of the two standards are identical; the characteristics of the elements in its original context (their form, required content, preconditions, objectives, post-conditions on use...) fully satisfy the requirements of the context in which it is to be reused.
- ***Partial map***: the elements are similar, but they are not identical; depending on the context and the objectives, the differences between them might be significant; in this case, a clear record of the similarities and differences is required.
- ***No map***: there is insufficient similarity between the elements to enable the user to assert a mapping; in this case, it may be important to record the differences, and the reasons why the mapping makes no sense, in order to spawn further a gap analysis and prevent inadvertent reuse.

Full maps are usually rare in the assurance domain and the majority of maps are partial.

Three elements play a role in equivalence mapping: artefacts, activities, and requirements. Pragmatically, any of these referenced assurable elements can be reused. The acceptability of the reuse needs to be argued in terms of the overall assurance objectives indicated by a standard, i.e. what needs to be demonstrated for assurance and compliance in the target context.

Equivalence maps are also necessary between the Baseline of an assurance project and the Reference Framework (or frameworks) of the standard according to which a system has to be assured. Some differences might exist as result of e.g. having to tailor how to follow a standard according to the specific characteristics of a system.

3) Compliance mapping

Another necessary type of map for cross-standard reuse is compliance map. These mappings specify how the information of an assurance project (i.e., its body of assurance assets) complies with its Baseline. As for equivalence maps, compliance maps can be full, partial, or no map. By mapping an artefact to a reference artefact selected for a baseline, the intent of the artefact must be indicated.

The compliance maps of the source assurance project will typically be full and 1:1. Its Baseline will correspond to a template, which is used in the project. For example, a Baseline from a Reference Framework for DO-178C will have Software Requirements Data as an artefact to provide, and an assurance project can have a single artefact that maps to Software Requirements Data. Nonetheless, an assurance project can also manage, structure, or group its artefacts in a different way as a standard indicates, but still being compliant. For example, an assurance project could have more than one artefact for its Software Requirements Data, such as high-level requirements specification and low-level requirements specification. Each of these artefacts would partially map to Software Requirements Data.

Compliance maps for the target assurance project can be derived from the compliance maps of the source project. In this case, the likelihood of derived full maps is low because of the differences that usually exist between standards. The assurance information of the source project fulfils the requirements of its baseline and this baseline has been generated from a specific Reference Framework (both belonging to the source project for the cross-standard reuse). The target assurance project will have a different Baseline and Reference Framework, thus different requirements to fulfil. Nonetheless, some reference requirements can be similar or equal. For example, an artefact that complies with EN 50128 Software Requirements Specification will partially map to the DO-178C Software Requirements Data.

4) Needs and gaps from certification artefact reuse

Finally, reuse of safety certification artefacts across standards and across domains requires that compliance needs and gaps resulting from the reuse are determined. This is a consequence of the fact that the maps between standards are most often partial. Compliance with a standard will allow a system to comply with another to some extent, but fulfilment of further reference requirements for the target project may be necessary. This will usually involve the execution of some additional activity and possibly the creation of some additional artefacts.

For example, reusing EN 50128 Software Requirements Specification in an avionics assurance project will lead to the partial fulfilment of the reference requirements for DO-178C Software Requirements Data. The corresponding compliance gaps must be filled for the achievement of full compliance with DO-178C.

3.2.2.1 Process for reuse of assurance assets across standards and domains

This process consists of seven main activities and it is based on the conceptual framework shown in Figure 27. This framework corresponds to excerpts of all the meta-models created in AMASS. Although presented as a sequence, some activities of the process can be executed in parallel, iteratively, or in a different order. For example, the Baseline of an assurance project can be modified while collecting new assurance assets because of some new requests by a safety assessor.

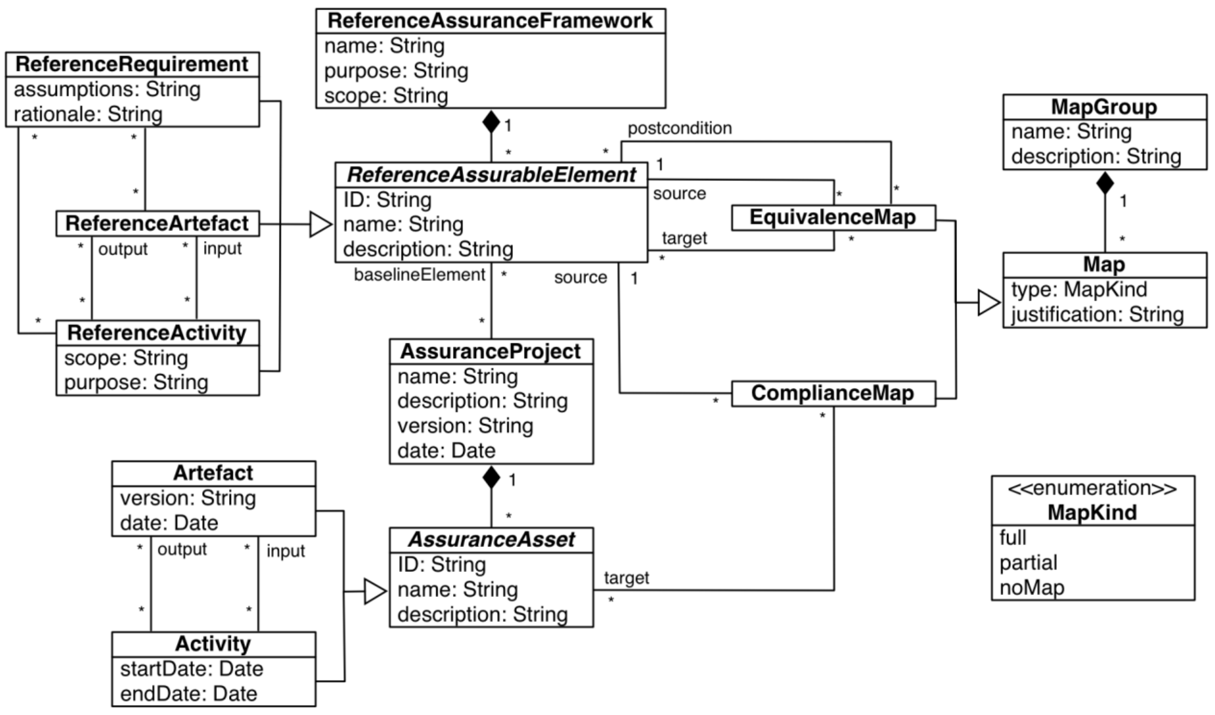


Figure 27. OpenCert reuse conceptual framework

The seven main activities are described below.

1) Create reference assurance frameworks

The first activity to be executed to apply the reuse approach is to specify the compliance needs of the standards for the source and the target domains. A reference assurance framework must be created for each standard, and the frameworks will be later used for specifying baselines and equivalence maps.

The Reference Framework will contain reference requirements to fulfil, reference activities to execute, and reference artefacts to manage. Reference artefacts can be linked to reference requirements and to reference activities (input/output), which enables the specification of reference artefact intents. In EN 50128 [21], Software Design Specification is a reference artefact, Component Design is a reference activity, and “test cases and their results shall be recorded, preferably in machine readable format for subsequent analysis” (clause 7.6.4.5.a) is a reference requirement for the Software Integration Test Report. Reference artefacts, activities, and requirements can be decomposed into others.

A reference framework can further contain information about the reference roles that might be involved in a safety/security-critical system’s lifecycle (e.g., designer), about the reference techniques that might be used to execute reference activities and create reference artefacts (e.g., formal methods), and about 1the applicability of the above elements (e.g., a given reference technique can be recommended for a given SIL in EN 50128). A reference artefact can also have reference artefact attributes (e.g., test outcome; passed or failed) and be linked to other reference artefacts by means of reference artefact relationships (e.g., Design Description satisfies Software Requirements Data).

2) Specify equivalence maps between the reference assurance frameworks

Once reference assurance frameworks are created, equivalence maps can then be specified between their reference assurable elements in order to determine how similar the frameworks are. Each equivalence map will have a source and a target reference assurable element, and can have a textual justification, and post-conditions. The post-conditions correspond to additional reference assurance elements that must be taken into account when an artefact is reused.

When creating equivalence maps for safety certification artefact reuse (i.e., between reference artefacts), and as explained above, it is essential to analyse the intent of the reference artefact to decide upon the extent to which two reference artefacts are similar. Arguing about the equivalence between reference artefacts requires discussion of the similarity of the associated reference requirements and reference activities.

The equivalence mapping process might not result in 1:1 maps between e.g. single reference artefacts. A set of source reference artefacts might map as a whole to a single target reference artefact. It can also be necessary to map different types of reference assurance elements. For example, there may be cases where no mapping can be asserted between a reference requirement of the target reference assurance framework and any of the reference requirements of the source, but that the former reference requirement could be mapped to some reference artefact in the source. For the two cases described in this paragraph, the specification of a map justification is especially important to document why the corresponding maps have been specified.

Another aspect to take into account is map consistency. For example, if the reference requirements of a reference artefact only partially map to the reference requirements of another reference artefact, from a different reference assurance framework, then the map between the reference artefacts must be “partial”.

3) Determine the Baseline for the source assurance project

The specific compliance needs of the source assurance project must be specified. To this end, the applicable elements of a reference assurance framework must be selected. It is also possible to refine the elements to project-specific needs or to specify information that it is not in the reference assurance framework. For example, DO-178C [22] does not explicitly define roles for assurance projects, but such roles might have to be declared for a specific project. When refining a reference assurance framework into a Baseline, the suitability of the changes introduced for the corresponding assurance project, in relation to how the Baseline corresponds to the reference assurance framework, might needed to be justified. This is achieved by specifying equivalence maps between the baseline and the reference assurance framework.

4) Collect the assurance assets of the source assurance project

The information of the assurance assets of the source assurance project must be collected in order to demonstrate how the system safety or security has been assured. For reuse purposes, the main assurance assets to collect are the artefacts created during the system lifecycle. Other assurance assets include the information about the activities executed, the participants in the assurance project, the techniques used, and the argumentation claims.

5) Specify compliance maps between the assurance assets of the source assurance project and its baseline

Compliance maps must be specified for the source assurance project in order to indicate how the project meets its compliance needs. The assurance assets of the project are mapped (full or partial map, or no map) to its Baseline, and a justification for the map might be specified.

6) Determine the baseline for the target assurance project

This activity is the same as for activity (3) but for the target assurance project instead of the source one.

7) Reuse assurance assets from the source assurance project to the target one

Finally, assurance assets from the source assurance project are reused in the target one, for the Baseline specified in activity (6). The reuse can result in additional compliance needs for the target assurance project according to the post-conditions of the equivalence maps. These needs will be included in the Baseline of the target project.

Reuse of assurance assets results in the generation of compliance maps for the target assurance project. This is based on the chain of maps consisting of the compliance maps for the source assurance project and of the equivalence map between reference assurance frameworks, including possible maps between a Baseline and a parent reference assurance framework. If there is some partial map in the chain, then the resulting compliance map for the target assurance project will also be partial. It might also be necessary to address some post-conditions, and it is also essential to analyse the map justifications in order to determine what compliance needs must still be addressed in the target assurance project. When all the maps are full, instead, then the derived compliance map is supposed to be full too. If there is any ‘no map’, then the reuse is not advisable. This means in practical terms that no reuse is possible, unless at the expense of significant reverse engineering activities, almost as if it was a new project.

For assurance asset reuse between projects, mapping the reference requirements can suffice because the reuse consequences can be determined from the correspondence between reference requirements. These consequences can correspond to the need of assuring further reference requirements in the target assurance project.

3.2.2.2 Usage of Tools

To create Equivalence Maps, a tailored functionality has been developed in the OpenCert tool. To open it, a button called “Mapping Set” is available on the Properties form of the Reference Framework using the tree view editor (see Figure 28).

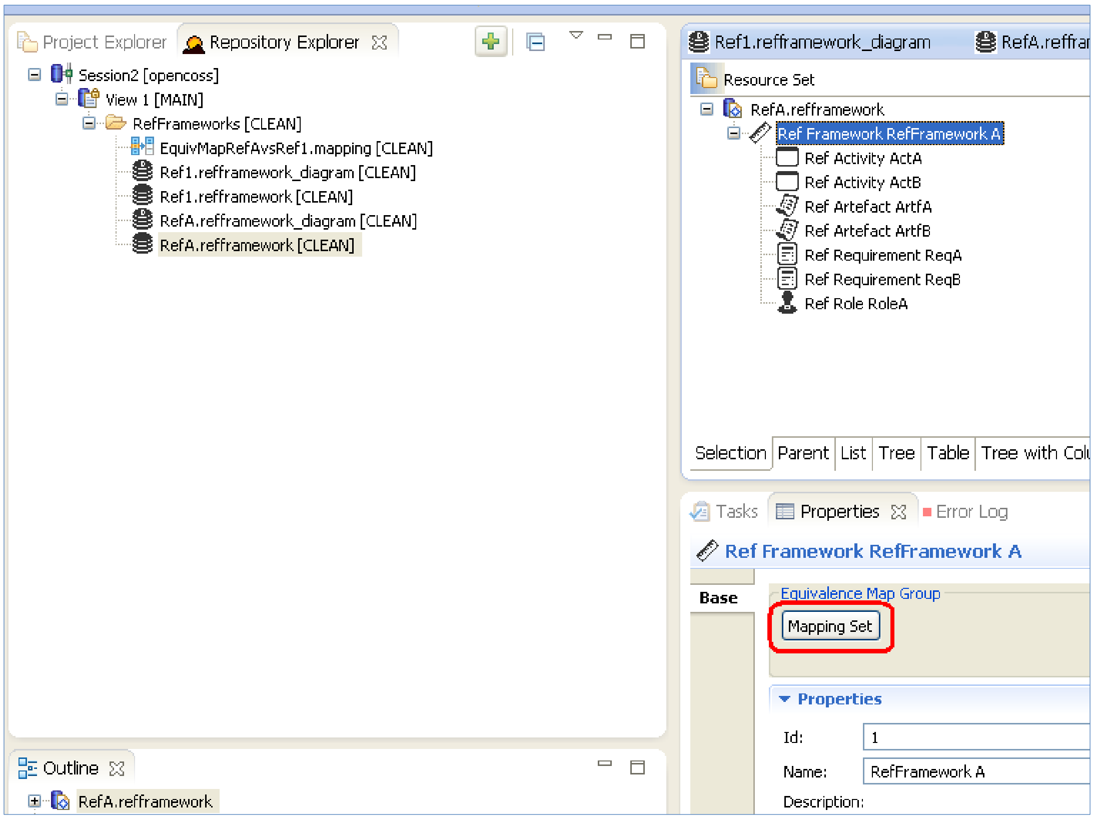


Figure 28. How to create Equivalence Maps

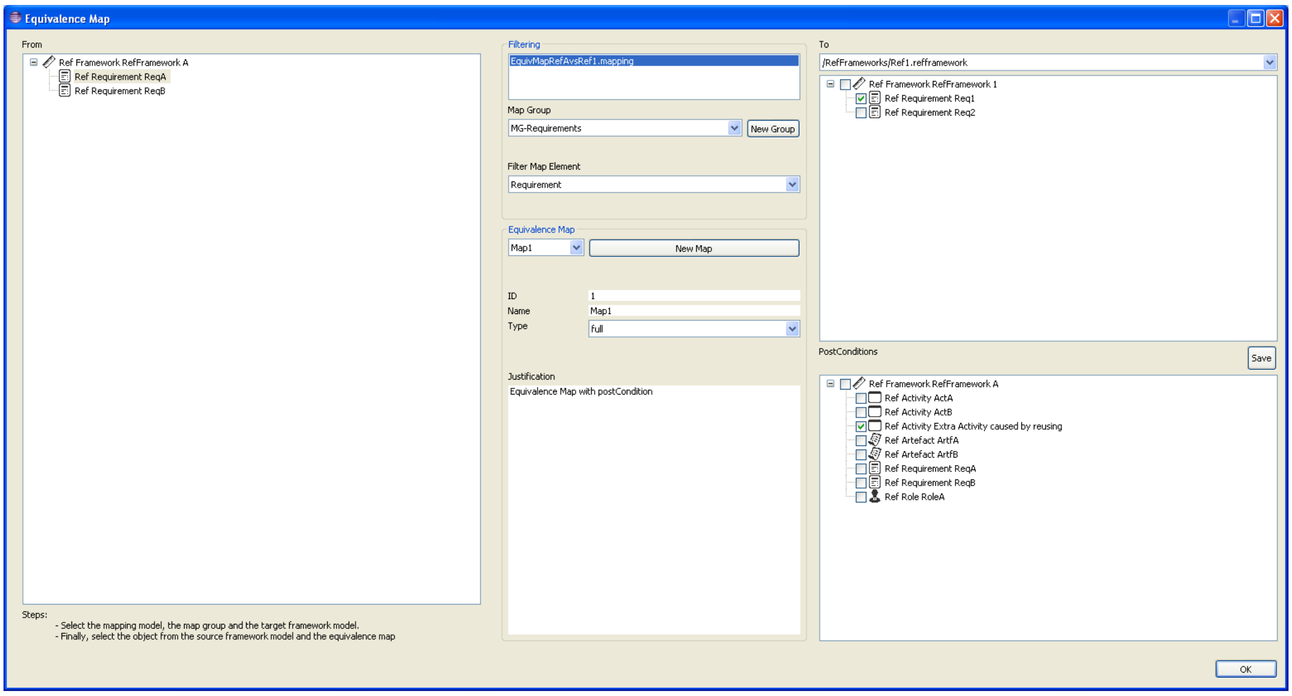
Figure 29 shows the form for Equivalence Map creation. The Equivalence Map form is organized in three zones:

- The *left zone* shows the actual reference framework (“From”) and loads the type of elements for which the user wants to make the equivalence maps.
- The *middle zone* allows to make different filters such as:

- *Filter Mapping Model,* that 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 in 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 and allows to select the Reference Framework that will be the targeted standard of the equivalence map ("To").
  - The *upper list* loads the elements, according to the filter selected, of the refframework chosen in the combo box that will be the targeted standard of the equivalence map.
  - The *lower list* displays the full content (not filtered) of the "From" refframework that will be post conditions in case of reusing. The post conditions are mandatory extra activities, not included in the standard that must be performed in case of reusing an asset element from another assurance project compliant with a different standard.


Equivalence Map

From
- Ref Framework RefFramework A
  - Ref Requirement ReqA
  - Ref Requirement ReqB

Steps:
- Select the mapping model, the map group and the target framework model.
- Finally, select the object from the source framework model and the equivalence map

Filtering: EquivMapRefAvsRef1.mapping
Map Group: MG-Requirements
Filter Map Element: Requirement

Equivalence Map: Map1
ID: 1
Name: Map1
Type: full
Justification: Equivalence Map with postCondition

To: /RefFrameworks/Ref1.refframework
- ☐ Ref Framework RefFramework 1
  - ✓ Ref Requirement Req1
  - ☐ Ref Requirement Req2

PostConditions
- ☐ Ref Framework RefFramework A
  - ☐ Ref Activity ActA
  - ☐ Ref Activity ActB
  - ✓ Ref Activity Extra Activity caused by reusing
  - ☐ Ref Artefact ArtfA
  - ☐ Ref Artefact ArtfB
  - ☐ Ref Requirement ReqA
  - ☐ Ref Requirement ReqB
  - ☐ Ref Role RoleA

Figure 29. Equivalence Map form

Figure 30 shows how to open the Reuse Assistant for cross-standard reuse. It's only possible the evidence reuse using this feature.

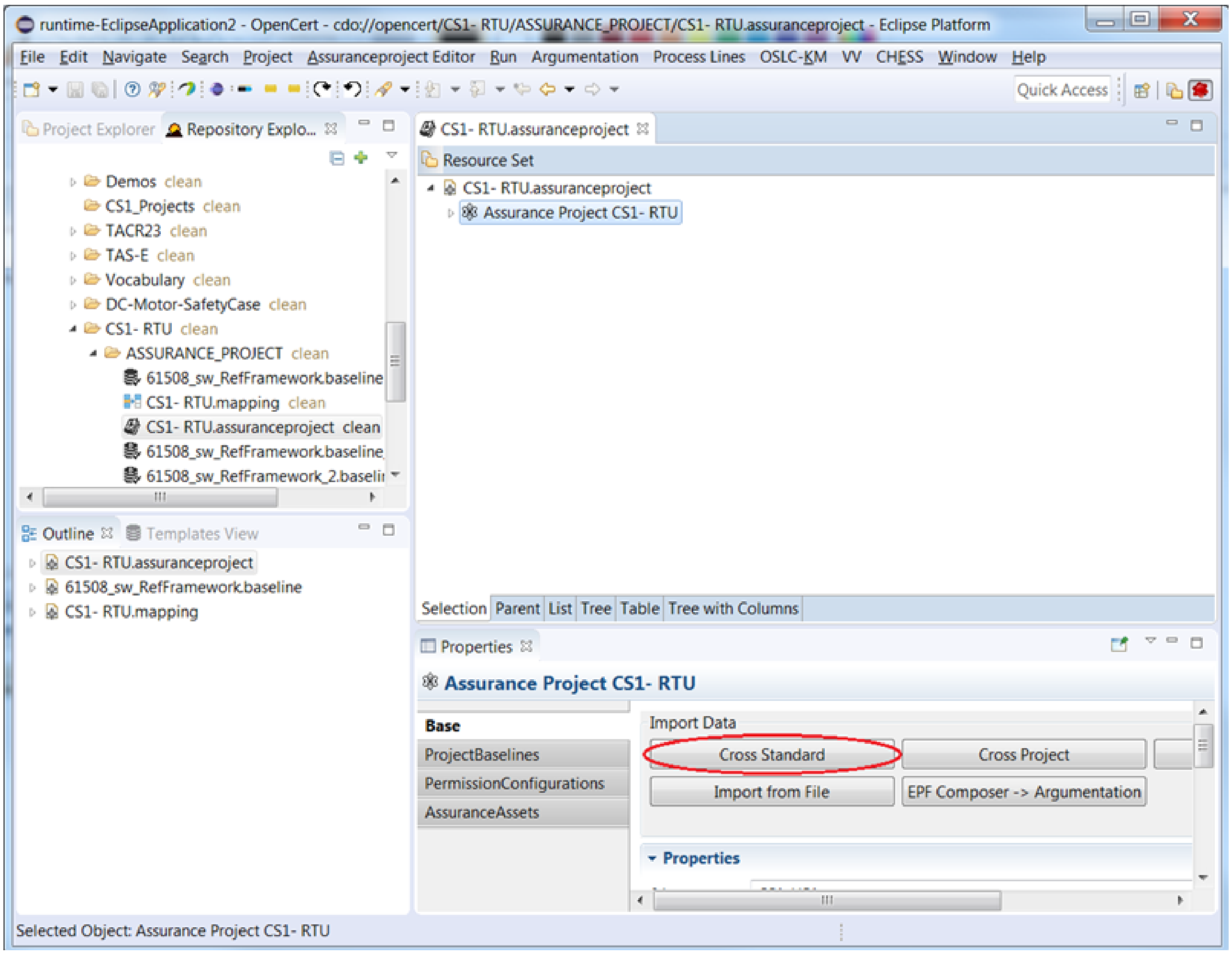
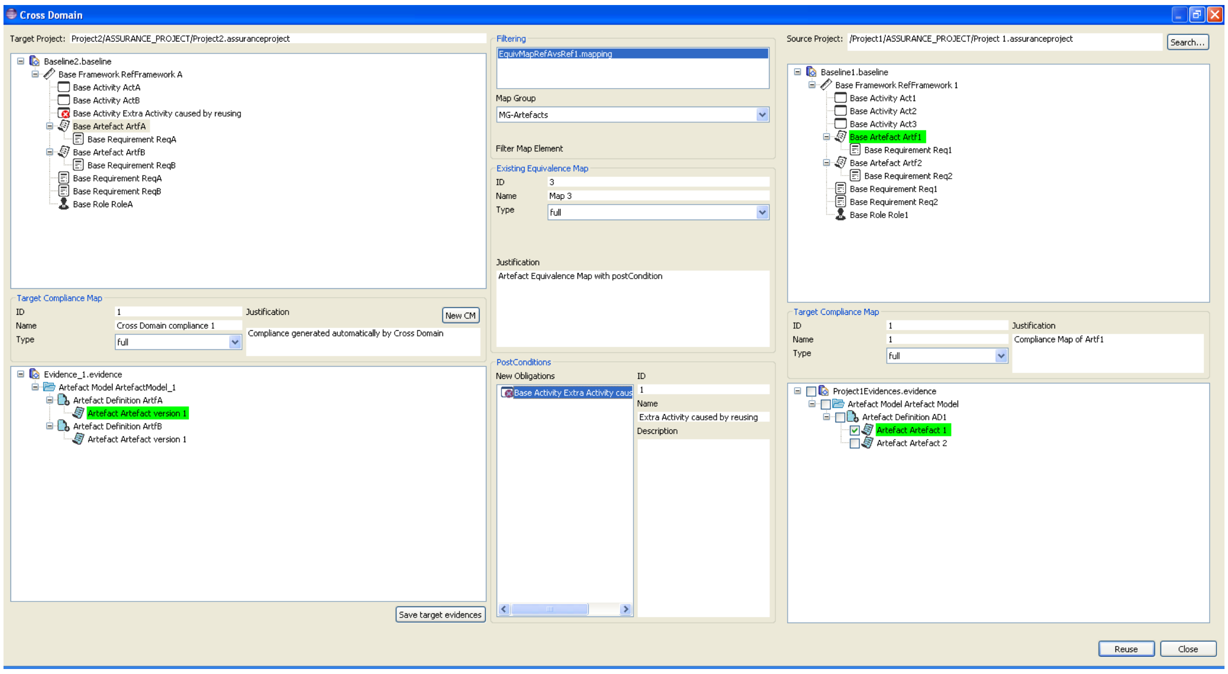

runtime-EclipseApplication2 - OpenCert - cdo://opencert/CS1- RTU/ASSURANCE_PROJECT/CS1- RTU.assuranceproject - Eclipse Platform
File Edit Navigate Search Project Assuranceproject Editor Run Argumentation Process Lines OSLC-KM VV CHESS Window Help
Quick Access
Project Explorer
Repository Explo...
Demos clean
CS1_Projects clean
TACR23 clean
TAS-E clean
Vocabulary clean
DC-Motor-SafetyCase clean
CS1- RTU clean
ASSURANCE_PROJECT clean
61508_sw_RefFramework.baseline
CS1- RTU.mapping clean
CS1- RTU.assuranceproject clean
61508_sw_RefFramework.baseline
61508_sw_RefFramework_2.baseli
Outline
Templates View
CS1- RTU.assuranceproject
61508_sw_RefFramework.baseline
CS1- RTU.mapping
CS1- RTU.assuranceproject
Resource Set
CS1- RTU.assuranceproject
Assurance Project CS1- RTU
Selection Parent List Tree Table Tree with Columns
Properties
Assurance Project CS1- RTU
Base
ProjectBaselines
PermissionConfigurations
AssuranceAssets
Import Data
Cross Standard
Cross Project
Import from File
EPF Composer -> Argumentation
Properties
Selected Object: Assurance Project CS1- RTU

Figure 30. Cross Standard button

Figure 31 shows the Cross Standard assistant form. It is organized in three zones:

- The *left zone* shows information about the target project. In the top part, the URL of the target assurance project, below a tree with the target baseline contents, below the compliance map information of the target baseline element selected and the contents of the target evidence model in another tree.
- The *middle zone* displays equivalence map information. It includes controls to select the equivalence mapping model and the equivalence map group and, to display the equivalence map details of the target baseline element selected and its post conditions in a list (to see the ID, Name and Description one post condition must be selected).
- The *right zone* presents information about the source project. In the top part, the URL of the source assurance project, below a tree with the source baseline contents, below the compliance map information of the source baseline element selected and the contents of the source evidence model.



Target Project: Project2/ASSURANCE_PROJECT/Project2.assuranceproject

- Baseline2.baseline
  - Base Framework RefFramework A
    - Base Activity ActA
    - Base Activity ActB
    - Base Activity Extra Activity caused by reusing
    - Base Artefact ArtfA
      - Base Requirement ReqA
    - Base Artefact ArtfB
      - Base Requirement ReqB
    - Base Requirement ReqA
    - Base Requirement ReqB
    - Base Role RoleA

Target Compliance Map
ID: 1
Name: Cross Domain compliance 1
Type: full
Justification: Compliance generated automatically by Cross Domain

- Evidence_1.evidence
  - Artefact Model ArtefactModel_1
    - Artefact Definition ArtfA
      - Artefact Artefact version 1
    - Artefact Definition ArtfB
      - Artefact Artefact version 1

Filtering
EquivMapRefAvsRef1.mapping
Map Group: MG-Artefacts
Filter Map Element

Existing Equivalence Map
ID: 3
Name: Map 3
Type: full
Justification: Artefact Equivalence Map with postCondition

PostConditions
New Obligations: Base Activity Extra Activity caus
ID: 1
Name: Extra Activity caused by reusing
Description

Source Project: /Project1/ASSURANCE_PROJECT/Project 1.assuranceproject

- Baseline1.baseline
  - Base Framework RefFramework 1
    - Base Activity Act1
    - Base Activity Act2
    - Base Activity Act3
    - Base Artefact Artf1
      - Base Requirement Req1
    - Base Artefact Artf2
      - Base Requirement Req2
    - Base Requirement Req1
    - Base Requirement Req2
    - Base Role Role1

Target Compliance Map
ID: 1
Name: 1
Type: full
Justification: Compliance Map of Artf1

- Project1Evidences.evidence
  - Artefact Model Artefact Model
    - Artefact Definition AD1
      - ✓ Artefact Artefact 1
      - Artefact Artefact 2

Figure 31. Cross standard window

The user must follow the following steps:

- Choose the source project of the reuse using the "Search" button (on the top right corner), so that the source baseline and evidence model tree will be loaded.
- Select the equivalence model and the equivalence group. An equivalence group serves to define specific map sets such as for example the equivalence maps of ISO 26262 [20] to Automotive Spice [48]. Another group can be the equivalence maps of ISO 26262 to IEC 61508 [62].
- 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 post conditions will be selected in the target Baseline model.

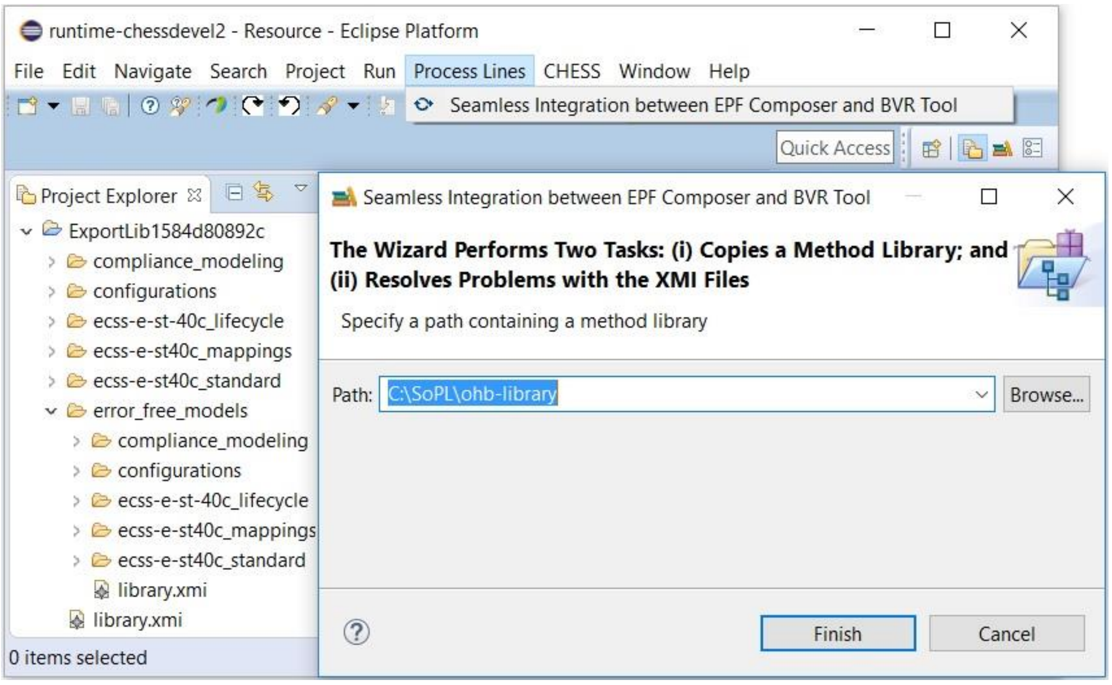
3.3 Management of Families/Lines

3.3.1 Process-related reuse via management of process lines

The predefined and standardized industrial processes tend to be reused, modified and extended to individual projects in a similar manner to the product lines. In this regard, the integration between process engineering and variability management activities is required. The conceptual design for management of families/lines is presented in deliverable D6.3 [12].

EPF-C offers a powerful solution to engineer processes. It stores all method library contents in a repository of XML Metadata Interchange (XMI). Specifically, the library contents are persisted in their own folders and XMI files. However, the XMI files produced by the EPF-C cannot be directly opened. The problems preventing the opening of EPF-C models needed to be resolved for mapping the elements of a target configuration and variability abstractions in BVR.

To perform the first two steps in Figure 34, a dialogue wizard has been implemented, as shown in Figure 32. 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) copies the contents of the method library into 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 Wizard Performs Two Tasks: (i) Copies a Method Library; and (ii) Resolves Problems with the XMI Files

Specify a path containing a method library

Path: C:\SoPL\ohb-library

Figure 32. The achievement of error free models in EPF-C

BVR-T offers a powerful solution to manage variability. The generation of target configurations with BVR-T is performed with three editors: VSpec, Resolution, and Realization. Variability modelling is performed in VSpec that extends Feature-Oriented Domain Analysis (FODA), usually called Feature Models, by including additional concepts such as variables, references and multiplicities. The VSpec model represents the tree structure having logical constraints to be considered during the resolution, as shown in Figure 33.

The mandatory features in VSpec are connected to the parent feature via solid lines, whereas the dashed lines represent optionality. The whole tree cannot be visualized due to space limitations; therefore, the minimize option (+) is used for hiding the features. The tasks associated with *software integration test plan development*, and *software units and software component integration and testing* are marked as optional. The multiplicity xor(1..1) is assigned to the criticality; therefore exactly one out of A, B, C and D must be selected for the software product. As the choices are associated with multiple criticality levels, the constraints have been applied; valid tailoring is guaranteed if the constraints are properly specified. For instance, the constraint *(A or B) implies (not Software_integration_test_plan)* indicates that the *Software_integration_test_plan* must be excluded for processes with criticality A or B. Likewise, the constraint *C implies (Software_integration_test_plan and ((not SUITP_K9) and (not SUITP_K10)))* enforces inclusion of *Software_integration_test_plan*, but also exclusion of SUITP K.9 and K10 for criticality C.

The Resolution editor permits variability engineers to perform the resolution and obtain resolved models. The resolution models are automatically generated from the VSpec model. However, to perform the resolution, engineers must specify the desired inclusion/exclusion choices for the specific configuration/resolution, as illustrated in Figure 34. The validation process is executed to conform whether the resolution corresponds to the VSpec model. It is possible to define multiple resolutions for the processes with variabilities.

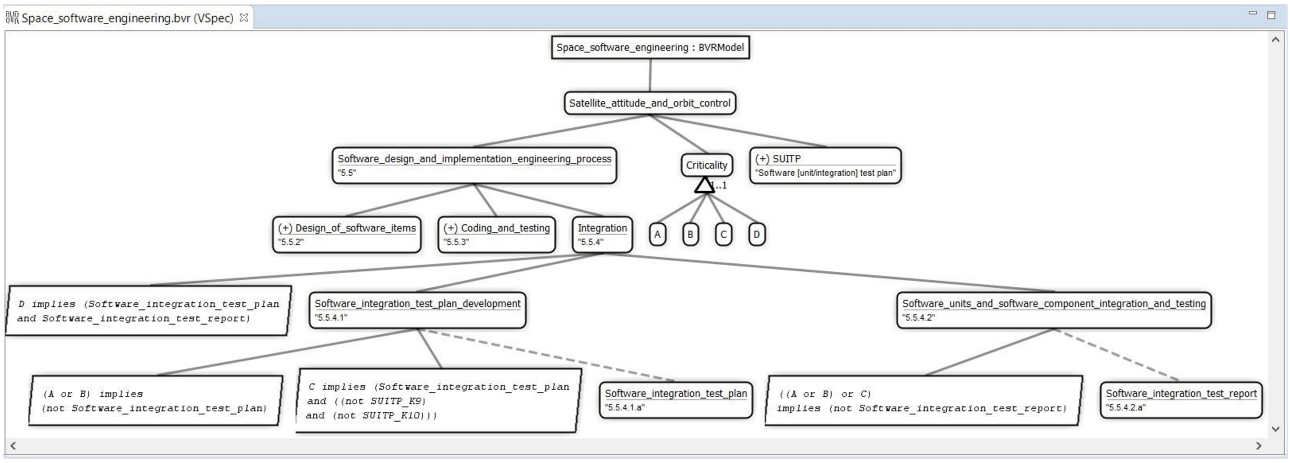

Space_software_engineering.bvr (VSpec)
Space_software_engineering : BVRModel
Satellite_attitude_and_orbit_control
Software_design_and_implementation_engineering_process "5.5"
Criticality
(+) SUITP "Software [unit/integration] test plan"
1..1
(+) Design_of_software_items "5.5.2"
(+) Coding_and_testing "5.5.3"
Integration "5.5.4"
A
B
C
D
D implies (Software_integration_test_plan and Software_integration_test_report)
Software_integration_test_plan_development "5.5.4.1"
Software_units_and_software_component_integration_and_testing "5.5.4.2"
(A or B) implies (not Software_integration_test_plan)
C implies (Software_integration_test_plan and ((not SUITP_K9) and (not SUITP_K10)))
Software_integration_test_plan "5.5.4.1.a"
((A or B) or C) implies (not Software_integration_test_report)
Software_integration_test_report "5.5.4.2.a"

Figure 33. VSpec model

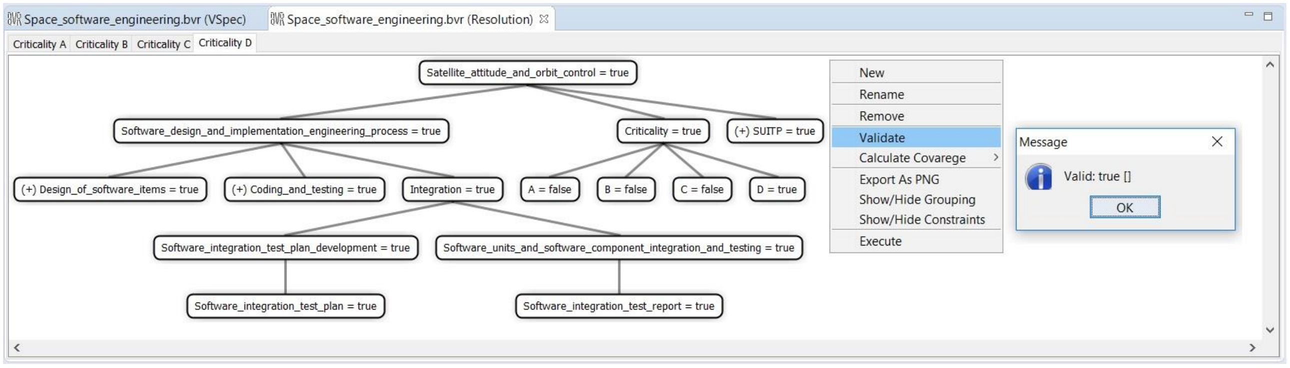

Space_software_engineering.bvr (VSpec)
Space_software_engineering.bvr (Resolution)
Criticality A
Criticality B
Criticality C
Criticality D
Satellite_attitude_and_orbit_control = true
Software_design_and_implementation_engineering_process = true
Criticality = true
(+) SUITP = true
(+) Design_of_software_items = true
(+) Coding_and_testing = true
Integration = true
A = false
B = false
C = false
D = true
Software_integration_test_plan_development = true
Software_units_and_software_component_integration_and_testing = true
Software_integration_test_plan = true
Software_integration_test_report = true
New
Rename
Remove
Validate
Calculate Covarege
Export As PNG
Show/Hide Grouping
Show/Hide Constraints
Execute
Message
Valid: true []
OK

Figure 34. Resolution model

Realization is based on the placements and replacements within the fragment substitutions. Specifically, the derivation process involves the substitutions in which elements of a placement fragment are removed and elements of a replacement are injected. To create the placements and replacements, the elements from the models are dragged and dropped on the Realization editor, as shown in Figure 35, for which the method library, plugins, configurations, content descriptions and/or processes should be considered.

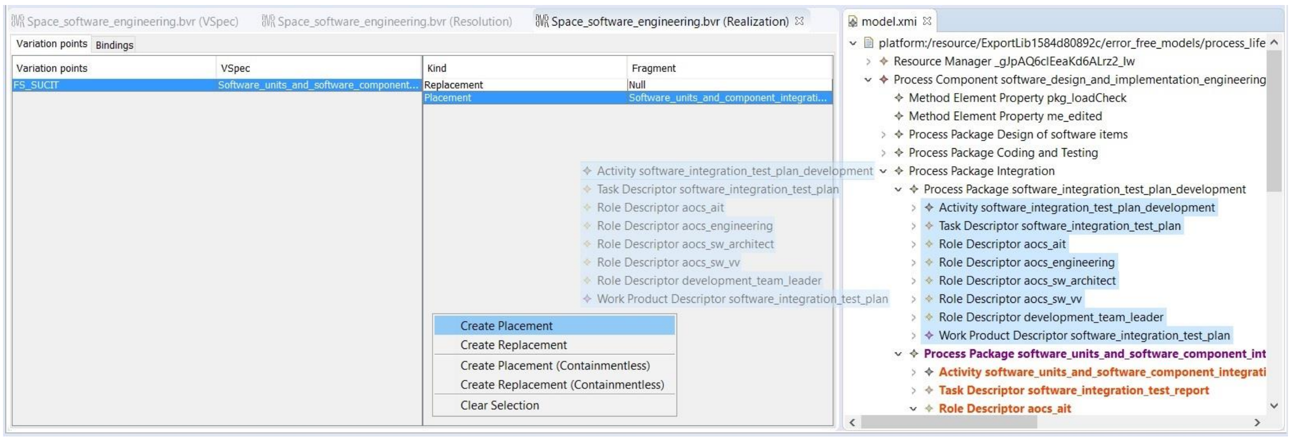

Space_software_engineering.bvr (VSpec)
Space_software_engineering.bvr (Resolution)
Space_software_engineering.bvr (Realization)
Variation points
Bindings
Variation points
VSpec
FS_SUCIT
Software_units_and_software_component...
Kind
Fragment
Replacement
Null
Placement
Software_units_and_component_integrati...
Activity software_integration_test_plan_development
Task Descriptor software_integration_test_plan
Role Descriptor aocs_ait
Role Descriptor aocs_engineering
Role Descriptor aocs_sw_architect
Role Descriptor aocs_sw_vv
Role Descriptor development_team_leader
Work Product Descriptor software_integration_test_plan
Create Placement
Create Replacement
Create Placement (Containmentless)
Create Replacement (Containmentless)
Clear Selection
model.xmi
platform:/resource/ExportLib1584d80892c/error_free_models/process_life
Resource Manager _gJpAQ6clEeaKd6ALrz2_lw
Process Component software_design_and_implementation_engineering
Method Element Property pkg_loadCheck
Method Element Property me_edited
Process Package Design of software items
Process Package Coding and Testing
Process Package Integration
Process Package software_integration_test_plan_development
Activity software_integration_test_plan_development
Task Descriptor software_integration_test_plan
Role Descriptor aocs_ait
Role Descriptor aocs_engineering
Role Descriptor aocs_sw_architect
Role Descriptor aocs_sw_vv
Role Descriptor development_team_leader
Work Product Descriptor software_integration_test_plan
Process Package software_units_and_software_component_int
Activity software_units_and_software_component_integrati
Task Descriptor software_integration_test_report
Role Descriptor aocs_ait

Figure 35. Realization editor

In order to derive the configuration, the execute option for the particular resolution is selected. This generates the target model, which is exported back to the EPF-C. If EPF-C 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 the “yes” option further supports the users in saving the copy of the previous model.

Pressing the “finish” button loads the derived process model in EPF-C. It shall be noted that the changes for resolving problems in XMI files and supporting the communication with the Realization editor had been reverted back in exported models.

3.3.1.1 Cross-concern variability management: an automotive SiSoPLE with EPF-C and BVR (*)

The management methodology for cross-concern variability is based on Security-informed Safety-oriented Process Line Engineering SiSoPLE [46], which can be used for intra and cross domain processes. It is part of an extensive process framework shown in Figure 13. The following provides a description concerning an intra domain and cross concern methodology. SiSoPLE is used to define an integrated process related to functional safety and cybersecurity. The intention of this approach (already described in D6.3 [12]) is to reuse elaborated process elements as much as possible.

Methodological variability management is a way to reuse such generically defined processes. It allows to instantiate activities according to defined parameters. The following specific process parameters are used: ASIL (Automotive Safety Integrity Level) for safety, SecRL (Security Risk Level) for security and RecL (Recommendation Level). The most benefit of process reuse provided by variability management is gained when process designers must deal with a lot of variable activities. In the AMASS Platform, tool support for process modelling is provided by EPF-C and variability management is covered by the BVR tool, which is used in different scenarios:

- Application of the methodology in the process definition phase. At this time, the value of the variability parameters is not known. For that reason, the process must be defined generically to enable later variations.
- Scenarios where process designers must deal with a lot of different project specific processes, which are based on identical standards. In this case, as many process elements as possible should be reused.

The process model based on underlying standards and company specific regulations is created in EPF-C. The expected input to the variability management tool is an EPF-C model and all relevant process parameters, which should be taken into consideration. BVR can define and evaluate any of these parameters. The output is an EPF-C model adjusted according to the selected input parameters.

The framework also deals with co-engineering, which is part of the process development activities, to define cross concern processes. BVR supports process developers to manage the variability of the integrated process. In our case the integrated process deals with functional safety and cybersecurity, two topics, which are executed in parallel, with high interaction. The process variability, more specific the parameter dependence of both topics, adds complexity, which needs a structured methodology to overcome. Co-engineering is a possible methodology to deal with highly interactive and variable activities like functional safety and cybersecurity.

A description concerning this part can be found in the deliverables D4.3 [8] and D4.8 [9]. The following paragraph describes a possibility to manage process variability.

From our point of view, process variability means that a process will be changed based on variability parameters like ASIL, SecRL and RecL. The created EPF-C model is directly related to the underlying standards and additional company specific regulations and activities. The mechanism to adapt activities and methods according to different parameters allows users to reuse processes within the range of these parameters. Based on the EPF-C model, which contains all available activities, a VSpec diagram, which covers variability concerns, must be defined in the BVR tool. This VSpec diagram contains only varying activities, which are related to the defined parameters ASIL, SecRL and RecL (see Figure 36). The BVR tool supports the creation of VSpec diagrams that are used by process developers to define which activities will vary and provides three supported variability types for modelling. In the available implementation, users must build the tree with variable activities manually. This means activities, parameters and methods from the EPF-C model are created manually in the VSpec diagram.

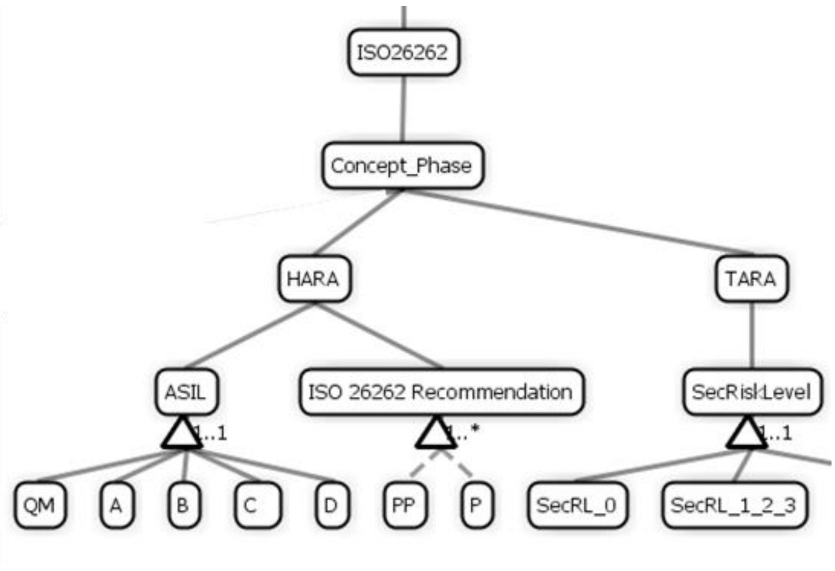


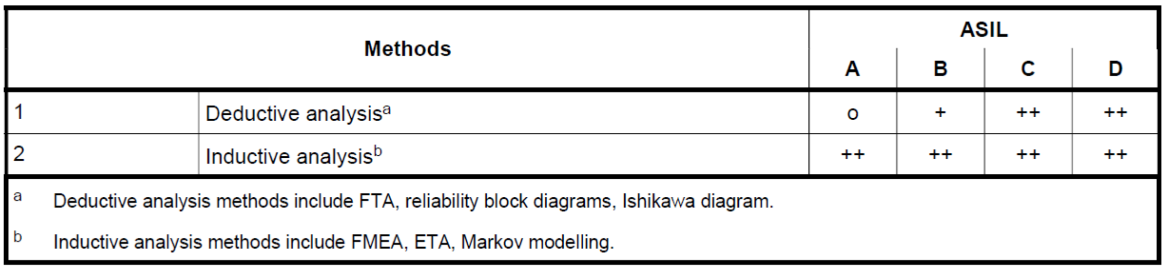
Figure 36. Vspec with different types of variability

Users can select between three different types of variability:

- None (0..*), none or any number of activities can be selected. This is used for optional activities.
- XOR (1..1), alternatives, one activity has to be selected exclusively. One activity is used instead of another.
- OR (1..*), one or more activities can be selected. At least one must be selected but as many as needed are possible.

The VSpec diagram also provides the possibility to use constraints, which are available to define which activities or methods must be performed under defined conditions. Constraints can be connected to activities to have a clear structure.

The constraints are based on Basic Constraint Language (BCL) grammar. The constraints must be defined in a way that the combination of the used parameters leads to the wanted and valid result. The selected values are checked with the constraints. The result of the evaluation of the constraints, done in the Resolution diagram, is "true" or "false". The result "true" means that the Resolution diagram is in accordance with its constraints. The values, which can be entered to an activity in the Resolution diagram, are as well "true" and "false". In the current implementation, the value "true" means that the activity will be replaced. The value must be "false" if the activity should be kept in the process.



Methods		ASIL			
		A	B	C	D
1	Deductive analysis[a]	o	+	++	++
2	Inductive analysis[b]	++	++	++	++

[a] Deductive analysis methods include FTA, reliability block diagrams, Ishikawa diagram.

[b] Inductive analysis methods include FMEA, ETA, Markov modelling.

Figure 37. Recommendation table from ISO 26262-4:2011

The following example shows how a constraint mechanism works in the BVR tool. The following parameters must be defined before they can be used in the BCL:

- Parameter ASIL: "C" indicates ASIL C
- Parameter RecL: "P" indicates the recommendation level plus (+) and "PP" indicates plus-plus (++).

Figure 37 shows an example for a recommendation taken from the ISO 26262-4 for Safety Analysis at System Level. For ASIL C, the safety process states that FTA is highly recommended, which is modelled by

ASIL="C" and RecL="PP". This means the standard demands FTA and it is prohibited to replace it. This prohibition means that the constraint must be "false", and the according expression is "(C and PP)) implies (not FTA)".

On the other hand, users can model a different ASIL B process, which deals only with "highly recommended" methods (ASIL = "B" and RecL = "PP"). The selection of the variant is done in the resolution diagram with help of the defined parameters shown in Figure 36. For the case "ASIL = B", ISO 26262 does not recommend FTA highly. It is only "recommended", which means that it is not necessary to execute this method. In this case, the constraint must be true to replace FTA. The according expression is: "(B and PP) implies FTA". To prevent a conflict with an ASIL B process, which includes "PP" and "P" activities, an additional term must be added. The complete expression is "(((B and PP) and (not P)) implies FTA".

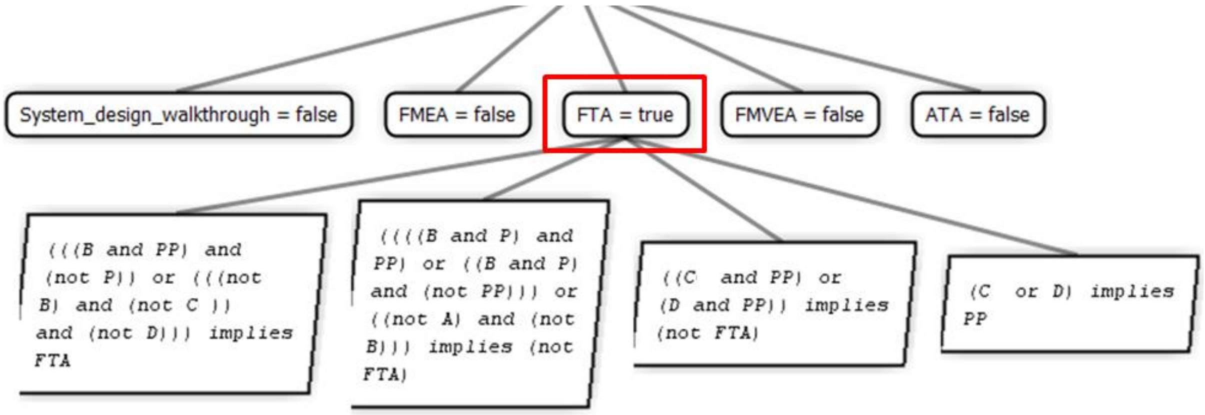

System_design_walkthrough = false
FMEA = false
FTA = true
FMVEA = false
ATA = false
(((B and PP) and (not P)) or (((not B) and (not C)) and (not D))) implies FTA
((((B and P) and PP) or ((B and P) and (not PP))) or ((not A) and (not B))) implies (not FTA)
((C and PP) or (D and PP)) implies (not FTA)
(C or D) implies PP

Figure 38. Constraints in the "Resolution diagram"

The example shows that it is difficult and time consuming to define all constraints manually because it happens easily that not all possible cases are noticed. A way, which is simpler and easy to automatize, is to add all possible combinations to the constraints. This may lead to a larger amount of terms but on the other hand you have a direct equivalence between recommendation table and derived constraints. A reduction may minimize the amount of expressions, but readers cannot retrace the link between recommendation table and constraints, which is demanded to follow the approach.

The Resolution diagram in the BVR tool supports the evaluation of the constraints. The diagram is generated automatically based on the VSpec diagram. Each choice, in our case a method (e.g. FTA) or an activity (e.g. System_design_walkthrough) in the tree has a changeable user defined Boolean value. These values are evaluated by the BVR tool. With help of the BVR function "validate", the tool checks all choices with the constraints. The result is "true" or "false". The result "true" means that the Resolution diagram is in accordance with its constraints and the process is correct. In this case the next step is the replacement of activities in the Realization diagram. The VSpec and the Resolution diagram define which activities are variable and which must be replaced. This replacement is done in the Realization diagram.

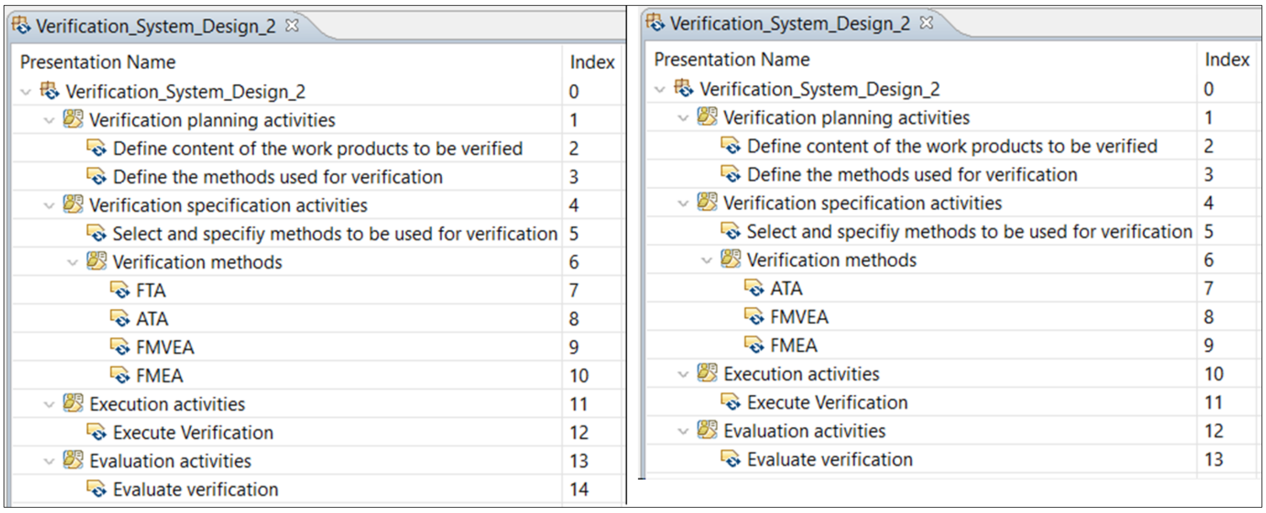
The Realization diagram is used to define "Placements" and "Replacements". A detailed description can be found in [37]. A placement defines the set of elements, which is to be replaced. The replacement is the new set, which is put to the placements position.

The initial step to create the Realization diagram is to import the associated EPF-C model, in the next step "Placements" and "Replacements" are defined. In our example, the replacement is always "null" because the intention is to remove elements from the process. The BVR-tool also supports exchange of activities. In this case the replacement is a different activity.

The next step is to connect placement and replacement with the Resolution diagram. This is done with the "FragmentSubstitution" in the Realization diagram. In the example, the placement "FTA" and the replacement "Null" are connected to the choice "FTA" in the Resolution diagram and the VSpec diagram.

Consequently, the placement in the VSpec diagram is replaced with "null" if the Boolean value of the element defined in the "FragmentSubstitution", in our case “FTA”, is "true".

The last step is the export of the changed process model. This means that the "execution" function of the Resolution diagram is used to export the final process model back to an EPF-C processable XMI format. Figure 39 shows a comparison between the EPF-C base process model (left side) and the changed BVR export process model (right side). After the export, the changed model is available in EPF-C and ready for execution by other tools like WEFACT [63].


Verification_System_Design_2

Presentation Name	Index
Verification_System_Design_2	0
Verification planning activities	1
Define content of the work products to be verified	2
Define the methods used for verification	3
Verification specification activities	4
Select and specifiy methods to be used for verification	5
Verification methods	6
FTA	7
ATA	8
FMVEA	9
FMEA	10
Execution activities	11
Execute Verification	12
Evaluation activities	13
Evaluate verification	14

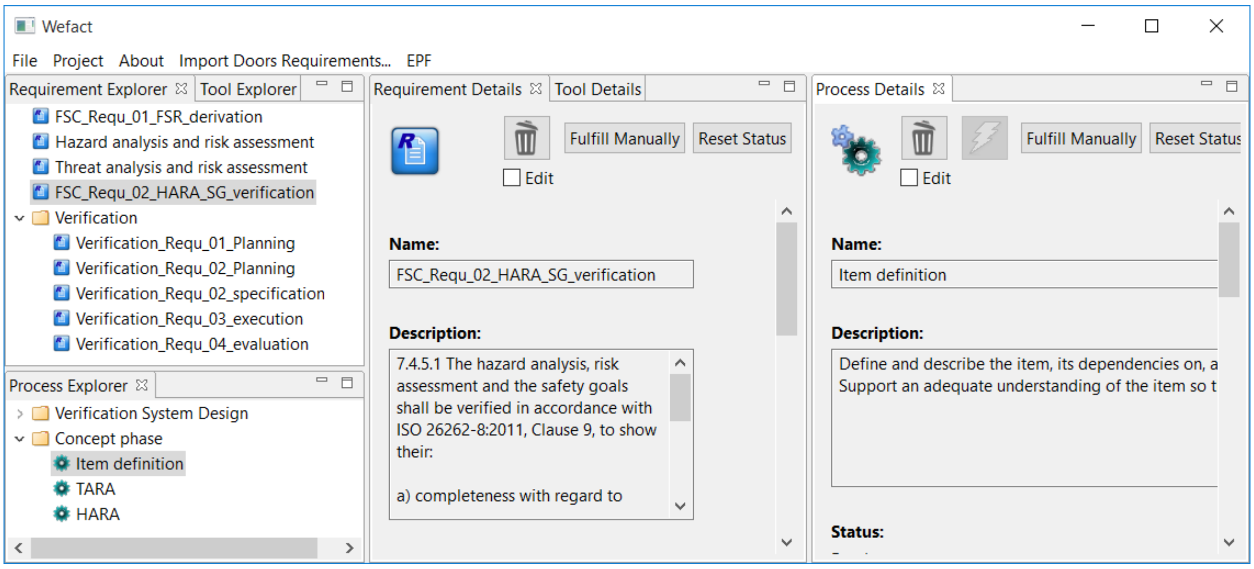
Verification_System_Design_2

Presentation Name	Index
Verification_System_Design_2	0
Verification planning activities	1
Define content of the work products to be verified	2
Define the methods used for verification	3
Verification specification activities	4
Select and specifiy methods to be used for verification	5
Verification methods	6
ATA	7
FMVEA	8
FMEA	9
Execution activities	10
Execute Verification	11
Evaluation activities	12
Evaluate verification	13

Figure 39. EPF-C model before (left) and after replacement of FTA (right)

Before the EPF-C model can be executed, it must be exported from EPF-C to an XML file and subsequently imported to WEFACT. In the next step requirements, input- and output files are linked to the process. Before the execution is performed, the workflow tools must be defined and associated with the process. These tools use the available input files and produce output files according to the process specification. The appearance of a new output file indicates that the process was executed successfully. The status of the activity in WEFACT changes to "successfully". WEFACT supports process execution activities, makes sure that requirements are fulfilled, related processes are executed properly, and all work products are available. The generated work product files are used as evidence in the assurance case.

A more detailed description concerning the process execution in WEFACT is available in the deliverable D4.3 [8].



FSC_Requ_01_FSR_derivation
Hazard analysis and risk assessment
Threat analysis and risk assessment
FSC_Requ_02_HARA_SG_verification
Verification
Verification_Requ_01_Planning
Verification_Requ_02_Planning
Verification_Requ_02_specification
Verification_Requ_03_execution
Verification_Requ_04_evaluation

Verification System Design
Concept phase
Item definition
TARA
HARA

Name: FSC_Requ_02_HARA_SG_verification

Description: 7.4.5.1 The hazard analysis, risk assessment and the safety goals shall be verified in accordance with ISO 26262-8:2011, Clause 9, to show their:

a) completeness with regard to

Name: Item definition

Description: Define and describe the item, its dependencies on, a
Support an adequate understanding of the item so t

Status:

Figure 40. Process related to verification in WEFACT (screenshot)

3.3.2 Product-related reuse via management of product lines (*)

The reuse-based development paradigms, such as component-based development and software product lines, have attracted considerable attention because of their potential benefits for reducing the development time, as well as the development and maintenance costs. In this context, the product related reuse strives for building a component once and re-use it in different applications or products. In the AMASS platform, the system components are modelled in CHESS [26]. A CHESS project (i.e. the folder where the model and other artefacts are stored) and a model can be created by using the CHESS dedicated Eclipse wizard ("**File → New → Other → CHESS…**"). The interested reader is referred to Section 7 of AMASS Platform User Manual [2] for further details about system component specification in CHESS Tool. The CHESSML [44] is implemented as UML, SysML and MARTE profile. But CHESSML does not offer any support for managing variability. The integration between component-based development and variability management activities is therefore required for configuring product families. Accordingly, the integration between CHESS and BVR-T has been achieved for supporting the variability management at component level.

The feature diagram associated to the component model is modelled in the VSpec editor (see Figure 42), the component configuration is performed in the Resolution editor (see Figure 43), and the placements and replacements are specified in the Realization editor (see Figure 44). The realization fragments have been specified for the .uml model. Fragment substitution removes elements within the placement and substitutes them with the elements in replacement. Therefore, the links between VSpec features and fragment substitutions need to be established. The inclusion/exclusion of particular choices for the configuration/resolution is therefore considered. For making specification intuitive and visual, the placements and replacements are highlighted in red and blue colours, respectively. In order to derive the configuration, the execute option in particular Resolution (editor) is selected. This generates the desired models that are automatically exported back to the CHESS project. The CHESS model is stored in .di, .notation and .uml files. In case of deletion from the .uml file, a command for removing orphan views from a given diagram is executed. For the replaced elements in .uml file, first the orphan views command is launched, and then dropping of replaced elements in the diagram(s) is performed.

3.3.2.1 Cross-domain variability management: an automotive/avionics product line (*)

This section provides the methodological guide for cross-domain product line. It is engineered from two different domain-specific products (an automotive unit and an avionics unit). The automotive unit by Infineon and the avionics unit by Lange are taken into consideration. CHESSML and BVR are used to model the cross-domain product line and to show how a CHESSML-compliant component model representing the architecture of one unit can be configured to obtain the CHESSML-compliant component model representing the architecture of the other unit.

The Lange aviation Central Computing (CCU) unit is depicted in Figure 41. Infineon automotive Electronic Control Unit (ECU) depicted in Figure 45. They share the same HW architecture.

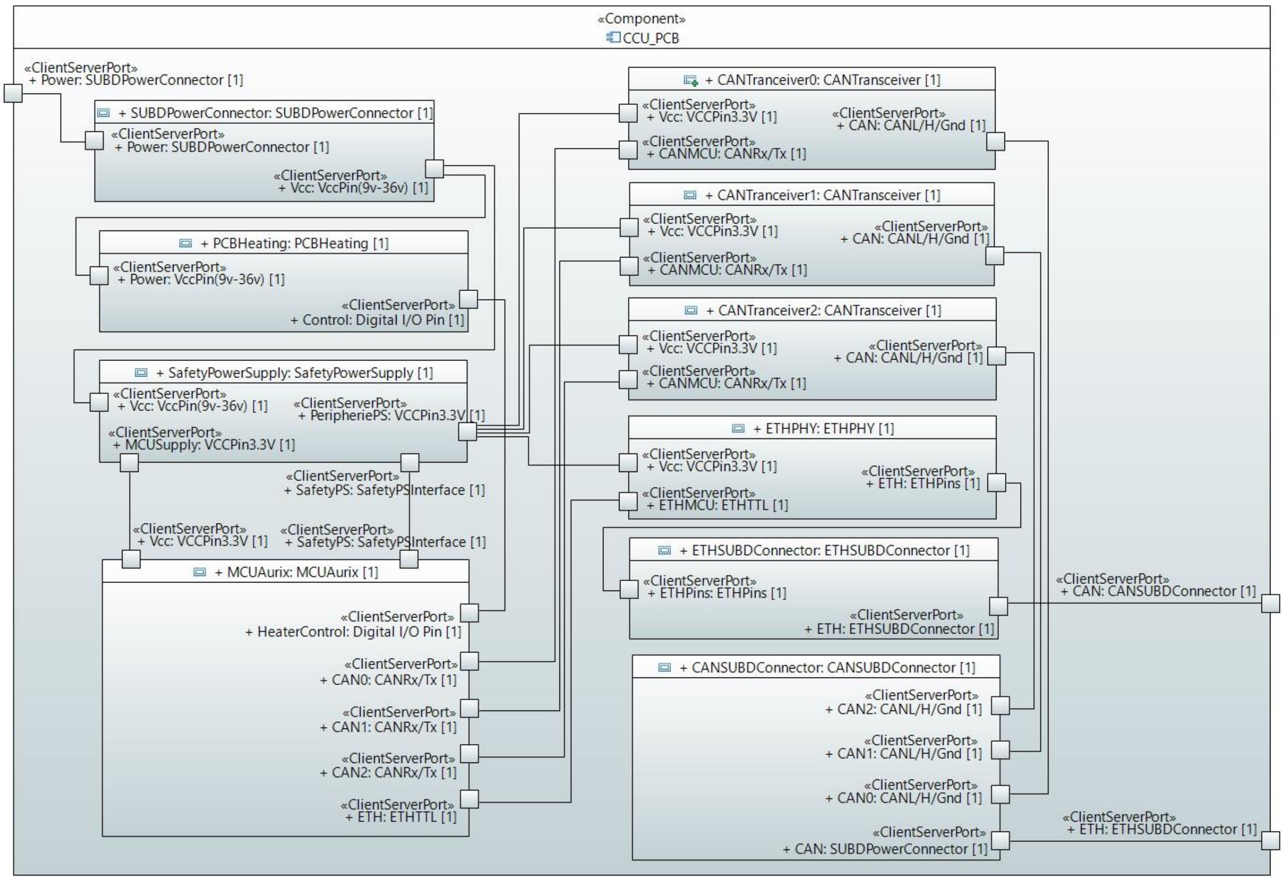

«Component»
CCU_PCB
«ClientServerPort» + Power: SUBDPowerConnector [1]
+ SUBDPowerConnector: SUBDPowerConnector [1]
«ClientServerPort» + Power: SUBDPowerConnector [1]
«ClientServerPort» + Vcc: VccPin(9v-36v) [1]
+ PCBHeating: PCBHeating [1]
«ClientServerPort» + Power: VccPin(9v-36v) [1]
«ClientServerPort» + Control: Digital I/O Pin [1]
+ SafetyPowerSupply: SafetyPowerSupply [1]
«ClientServerPort» + Vcc: VccPin(9v-36v) [1]
«ClientServerPort» + PeripheriePS: VCCPin3.3V [1]
«ClientServerPort» + MCUSupply: VCCPin3.3V [1]
«ClientServerPort» + SafetyPS: SafetyPSInterface [1]
«ClientServerPort» + Vcc: VCCPin3.3V [1]
«ClientServerPort» + SafetyPS: SafetyPSInterface [1]
+ MCUAurix: MCUAurix [1]
«ClientServerPort» + HeaterControl: Digital I/O Pin [1]
«ClientServerPort» + CAN0: CANRx/Tx [1]
«ClientServerPort» + CAN1: CANRx/Tx [1]
«ClientServerPort» + CAN2: CANRx/Tx [1]
«ClientServerPort» + ETH: ETHTTL [1]
+ CANTranceiver0: CANTransceiver [1]
«ClientServerPort» + Vcc: VCCPin3.3V [1]
«ClientServerPort» + CAN: CANL/H/Gnd [1]
«ClientServerPort» + CANMCU: CANRx/Tx [1]
+ CANTranceiver1: CANTransceiver [1]
«ClientServerPort» + Vcc: VCCPin3.3V [1]
«ClientServerPort» + CAN: CANL/H/Gnd [1]
«ClientServerPort» + CANMCU: CANRx/Tx [1]
+ CANTranceiver2: CANTransceiver [1]
«ClientServerPort» + Vcc: VCCPin3.3V [1]
«ClientServerPort» + CAN: CANL/H/Gnd [1]
«ClientServerPort» + CANMCU: CANRx/Tx [1]
+ ETHPHY: ETHPHY [1]
«ClientServerPort» + Vcc: VCCPin3.3V [1]
«ClientServerPort» + ETH: ETHPins [1]
«ClientServerPort» + ETHMCU: ETHTTL [1]
+ ETHSUBDConnector: ETHSUBDConnector [1]
«ClientServerPort» + ETHPins: ETHPins [1]
«ClientServerPort» + ETH: ETHSUBDConnector [1]
«ClientServerPort» + CAN: CANSUBDConnector [1]
+ CANSUBDConnector: CANSUBDConnector [1]
«ClientServerPort» + CAN2: CANL/H/Gnd [1]
«ClientServerPort» + CAN1: CANL/H/Gnd [1]
«ClientServerPort» + CAN0: CANL/H/Gnd [1]
«ClientServerPort» + CAN: SUBDPowerConnector [1]
«ClientServerPort» + ETH: ETHSUBDConnector [1]

Figure 41. Lange Aviation Central Computing Unit Model

The pure automotive components, such as LIN Transceiver cannot be re-used for aviation. The identified reusable components from automotive to aviation domain are: AURIX MCU, Safety Power Supply, automotive CAN Transceiver (several CAN are used, AURIX provides up to 6 CAN) and automotive Ethernet Transceiver. Besides the common components, the additional aviation-only components are required. For example, PCB Heating (required for -55°C environment qualification) is reuse enabler, without PCB Heating reuse of specified components is not possible. The connectors of ECU shall also be replaced with the aviation grade SUB-D connectors in CCU. The interested reader is referred to Sections 5.3.3.2.1 and 5.3.3.2.2 of D6.3 [12] for details about the components used in CCU and ECU.

In a similar manner to the process lines, the generation of target configurations is performed with VSpec, Resolution and Realization editors, as shown in Figure 42, Figure 43 and Figure 44, respectively.

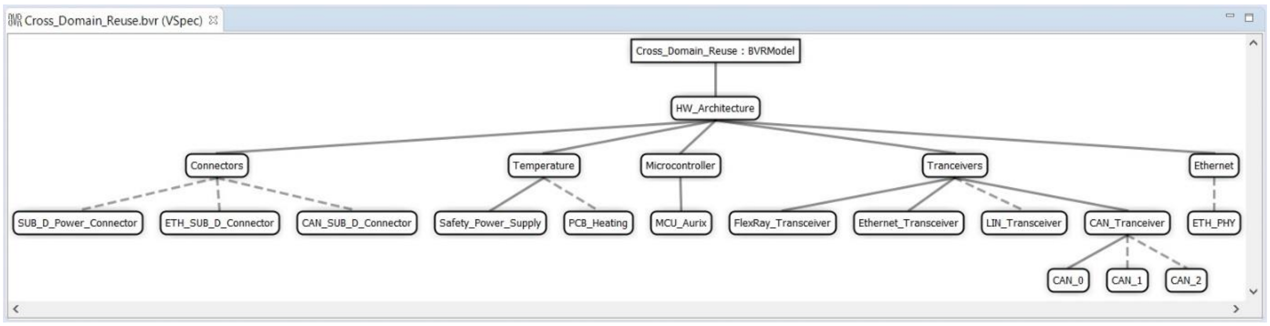


Figure 42. VSpec representing the cross-domain product line

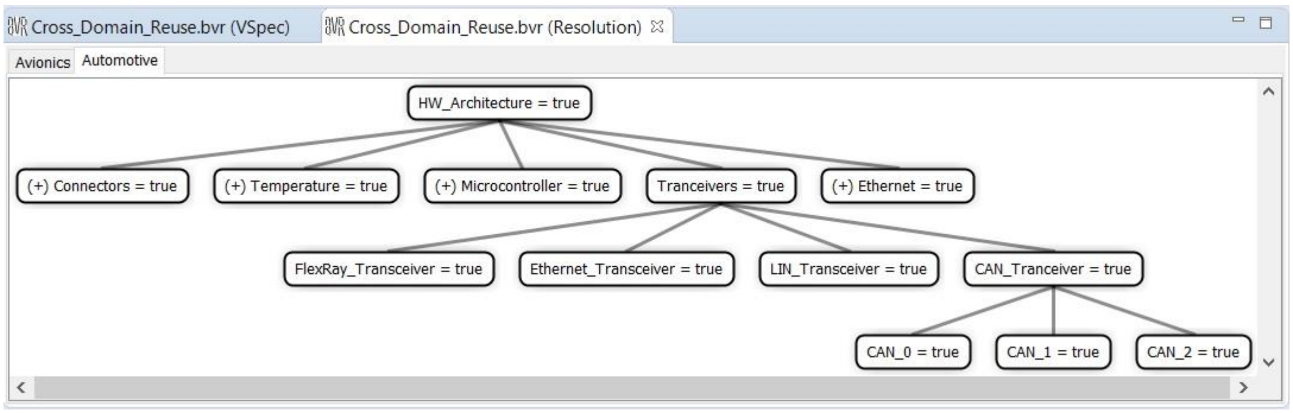
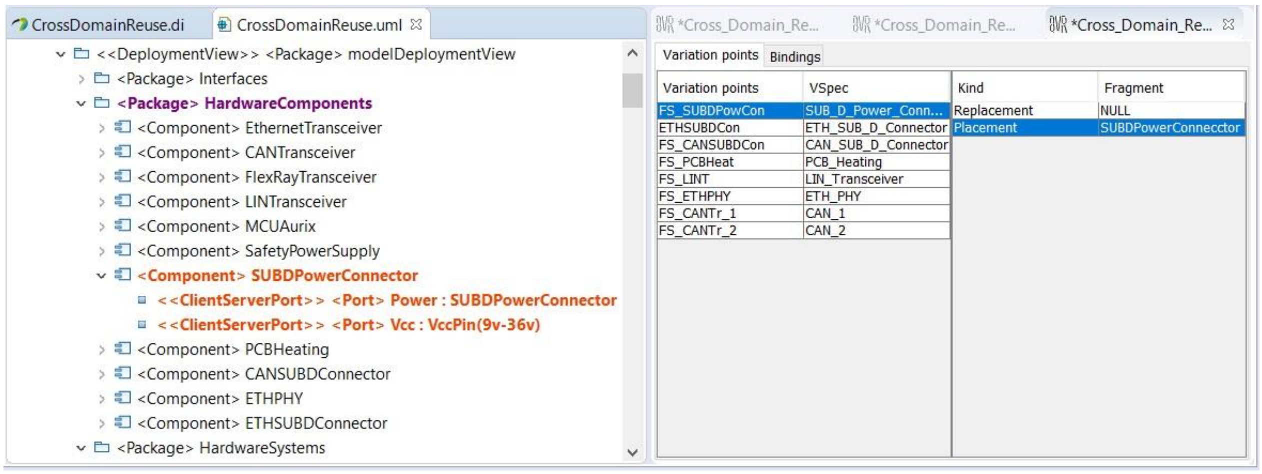


Figure 43. Variability Resolution for Automotive


CrossDomainReuse.di | CrossDomainReuse.uml

- <<DeploymentView>> <Package> modelDeploymentView
  - <Package> Interfaces
  - <Package> HardwareComponents
    - <Component> EthernetTransceiver
    - <Component> CANTransceiver
    - <Component> FlexRayTransceiver
    - <Component> LINTransceiver
    - <Component> MCUAurix
    - <Component> SafetyPowerSupply
    - <Component> SUBDPowerConnector
      - <<ClientServerPort>> <Port> Power : SUBDPowerConnector
      - <<ClientServerPort>> <Port> Vcc : VccPin(9v-36v)
    - <Component> PCBHeating
    - <Component> CANSUBDConnector
    - <Component> ETHPHY
    - <Component> ETHSUBDConnector
  - <Package> HardwareSystems

*Cross_Domain_Re... | *Cross_Domain_Re... | *Cross_Domain_Re...

Variation points | Bindings

Variation points	VSpec
FS_SUBDPowCon	SUB_D_Power_Conn...
ETHSUBDCon	ETH_SUB_D_Connector
FS_CANSUBDCon	CAN_SUB_D_Connector
FS_PCBHeat	PCB_Heating
FS_LINT	LIN_Transceiver
FS_ETHPHY	ETH_PHY
FS_CANTr_1	CAN_1
FS_CANTr_2	CAN_2

Kind	Fragment
Replacement	NULL
Placement	SUBDPowerConnecctor

Figure 44. Cross-domain Realization

The execution of Automotive resolution generates the desired model, as shown in Figure 43. The re-configured model and diagram are updated in the CHESS editor.

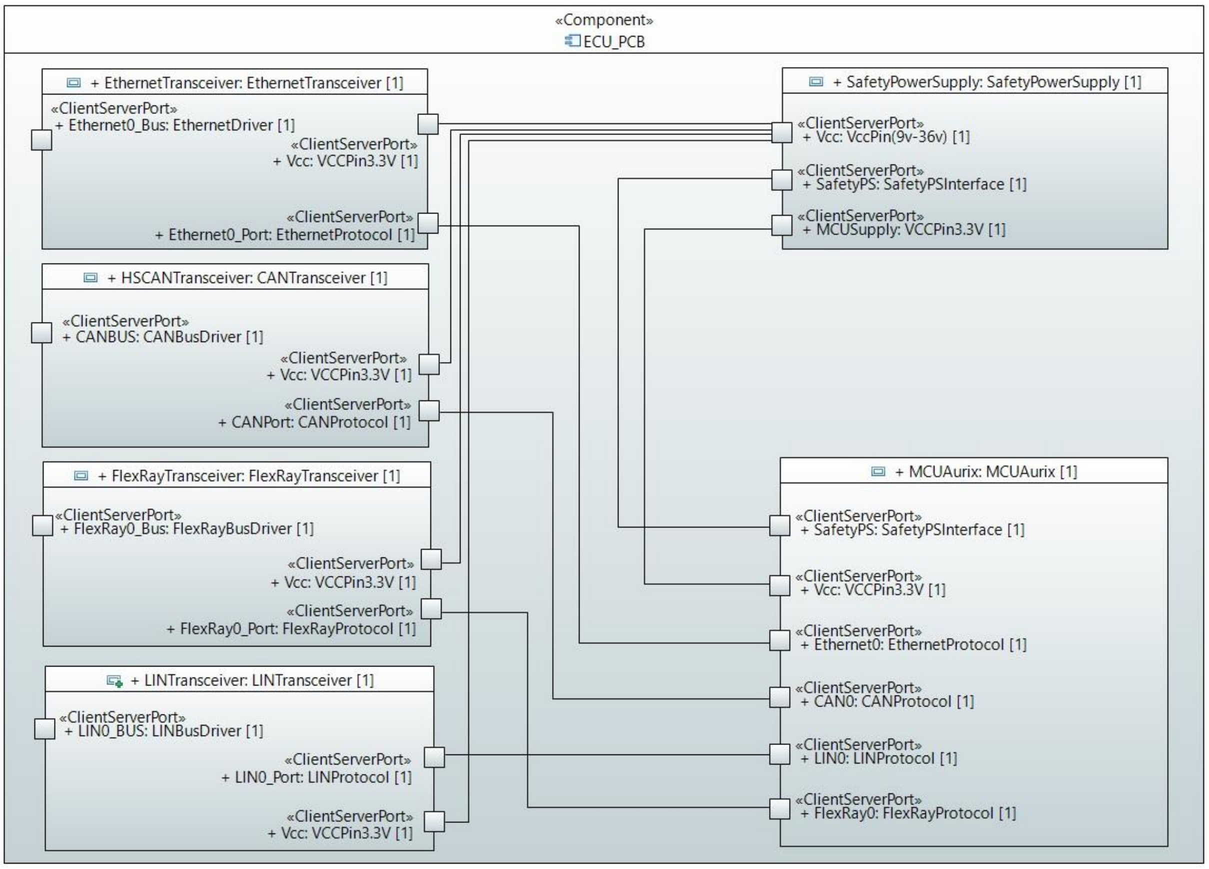

«Component»
ECU_PCB
+ EthernetTransceiver: EthernetTransceiver [1]
«ClientServerPort» + Ethernet0_Bus: EthernetDriver [1]
«ClientServerPort» + Vcc: VCCPin3.3V [1]
«ClientServerPort» + Ethernet0_Port: EthernetProtocol [1]
+ HSCANTransceiver: CANTransceiver [1]
«ClientServerPort» + CANBUS: CANBusDriver [1]
«ClientServerPort» + Vcc: VCCPin3.3V [1]
«ClientServerPort» + CANPort: CANProtocol [1]
+ FlexRayTransceiver: FlexRayTransceiver [1]
«ClientServerPort» + FlexRay0_Bus: FlexRayBusDriver [1]
«ClientServerPort» + Vcc: VCCPin3.3V [1]
«ClientServerPort» + FlexRay0_Port: FlexRayProtocol [1]
+ LINTransceiver: LINTransceiver [1]
«ClientServerPort» + LIN0_BUS: LINBusDriver [1]
«ClientServerPort» + LIN0_Port: LINProtocol [1]
«ClientServerPort» + Vcc: VCCPin3.3V [1]
+ SafetyPowerSupply: SafetyPowerSupply [1]
«ClientServerPort» + Vcc: VccPin(9v-36v) [1]
«ClientServerPort» + SafetyPS: SafetyPSInterface [1]
«ClientServerPort» + MCUSupply: VCCPin3.3V [1]
+ MCUAurix: MCUAurix [1]
«ClientServerPort» + SafetyPS: SafetyPSInterface [1]
«ClientServerPort» + Vcc: VCCPin3.3V [1]
«ClientServerPort» + Ethernet0: EthernetProtocol [1]
«ClientServerPort» + CAN0: CANProtocol [1]
«ClientServerPort» + LIN0: LINProtocol [1]
«ClientServerPort» + FlexRay0: FlexRayProtocol [1]

Figure 45. Infineon Automotive Electronic Computing Unit Model

3.3.3 Assurance case-related reuse via management of assurance case lines (*)

To support the variability management at assurance case level, the integration between OpenCert and BVR-T has been achieved. The generation of target models is performed with three editors: VSpec, Resolution, and Realization. In the assurance case lines, the commonality indicates the assurance case elements that do not vary and that characterize the family of assurance cases, while the variability indicates the assurance case elements that vary and that characterize the individuals within a family of assurance cases. Two kinds of variability might be identified within a set of assurance cases: (i) whenever there is more than one argumentation style to support the claims of a particular product-line instance (intrinsic); and (ii) whenever reusable assets referenced in the assurance case are bound to concrete assets within product-line models, such as the feature and reference architectural models vary (extrinsic). Figure 46 shows a fragment modelled by using the assurance case editor of OpenCert.

To create the argumentation model and diagram in a project, the dedicated wizard ("**File → New → Other → OpenCert → Arg Diagram to File**") is used. The separate creation of arg model and arg diagram is also supported. To initialize the arg_diagram from an arg model, the user needs to right-click at an arg model and select the "initialize arg_diagram diagram file" command. Then, the diagram elements can be dragged from the "Outline View" of Eclipse and dropped into the diagram editor. The "Arrange All" command can be executed. The interested reader is referred to Section 9 of user manual [2] for further details. Figure 47 illustrates the argumentation for FLEDS (Fuel Level Estimation and Display System) in OpenCert.

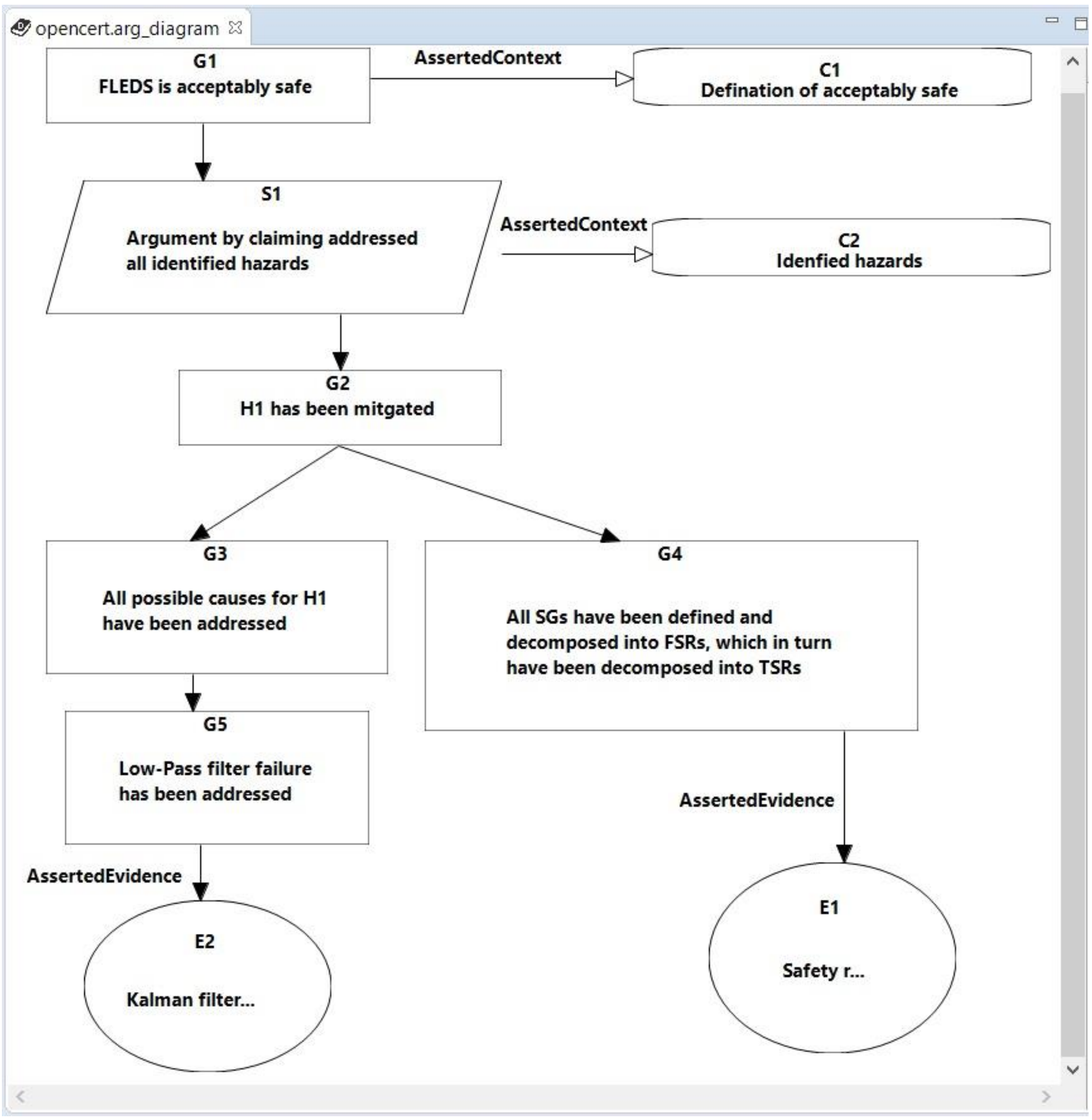

opencert.arg_diagram
G1
FLEDS is acceptably safe
AssertedContext
C1
Defination of acceptably safe
S1
Argument by claiming addressed all identified hazards
AssertedContext
C2
Idenfied hazards
G2
H1 has been mitgated
G3
All possible causes for H1 have been addressed
G4
All SGs have been defined and decomposed into FSRs, which in turn have been decomposed into TSRs
G5
Low-Pass filter failure has been addressed
AssertedEvidence
E2
Kalman filter...
AssertedEvidence
E1
Safety r...

Figure 46. Argumentation for FLEDS in OpenCert

The VSpec model is shown in Figure 47. It presents the tree structure with logical constraints to be checked during the resolution. G5 and G6 represent alternative argumentation fragments. For instance, the constraint “Truck implies G5” indicates that the choice G5 must be included in the resolution for Truck.

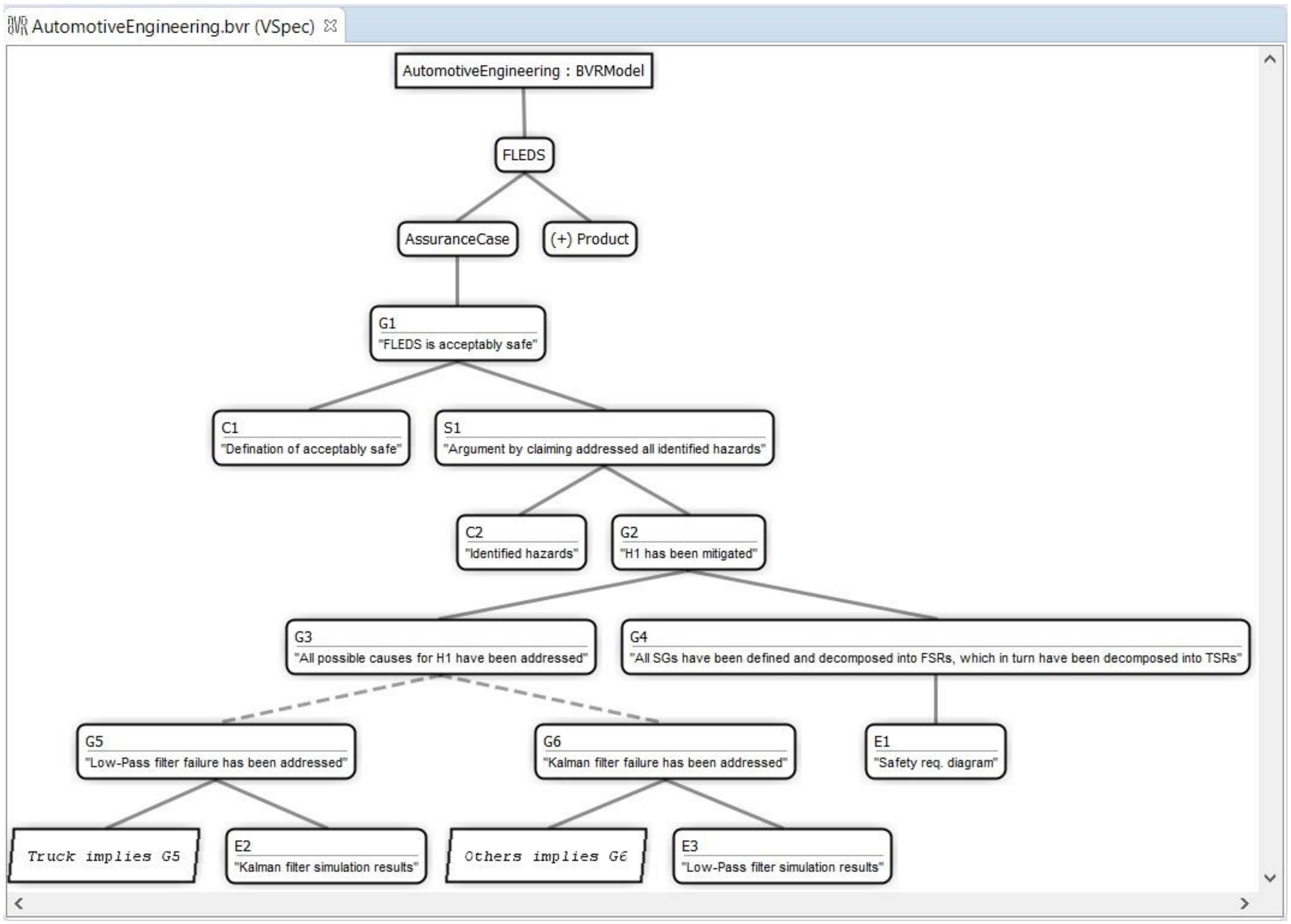
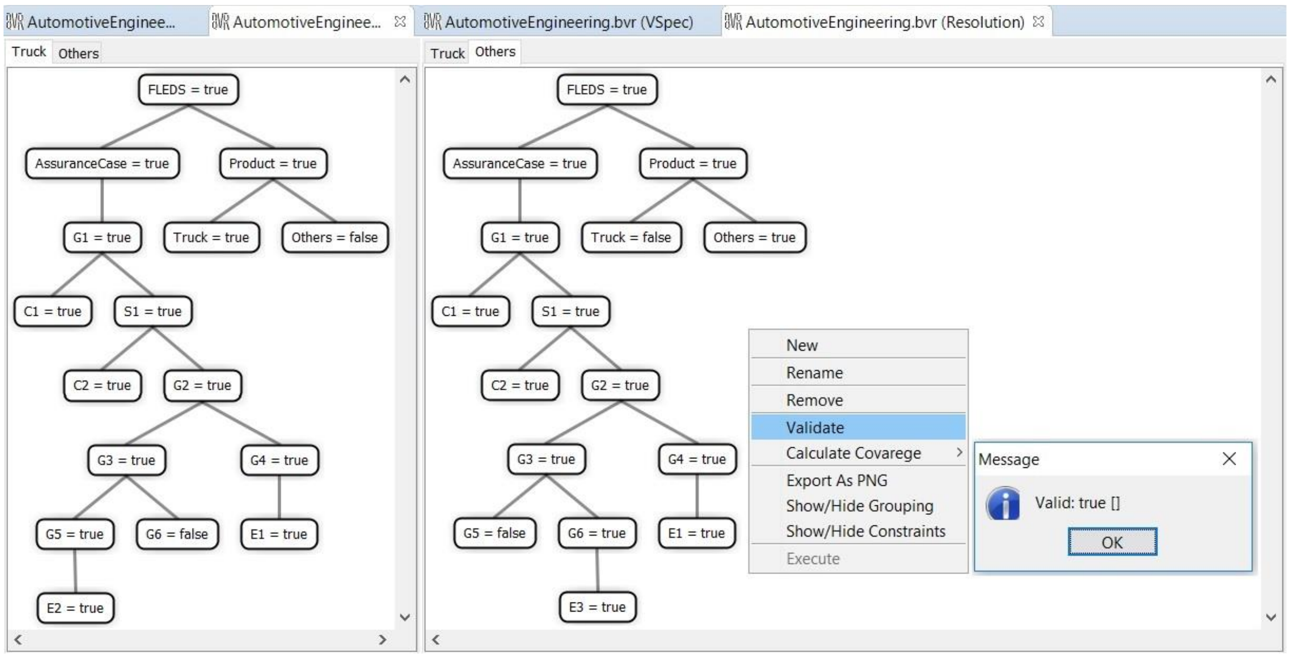

AutomotiveEngineering.bvr (VSpec)
AutomotiveEngineering : BVRModel
FLEDS
AssuranceCase
(+) Product
G1
"FLEDS is acceptably safe"
C1
"Defination of acceptably safe"
S1
"Argument by claiming addressed all identified hazards"
C2
"Identified hazards"
G2
"H1 has been mitigated"
G3
"All possible causes for H1 have been addressed"
G4
"All SGs have been defined and decomposed into FSRs, which in turn have been decomposed into TSRs"
G5
"Low-Pass filter failure has been addressed"
G6
"Kalman filter failure has been addressed"
E1
"Safety req. diagram"
Truck implies G5
E2
"Kalman filter simulation results"
Others implies G6
E3
"Low-Pass filter simulation results"

Figure 47. VSpec model

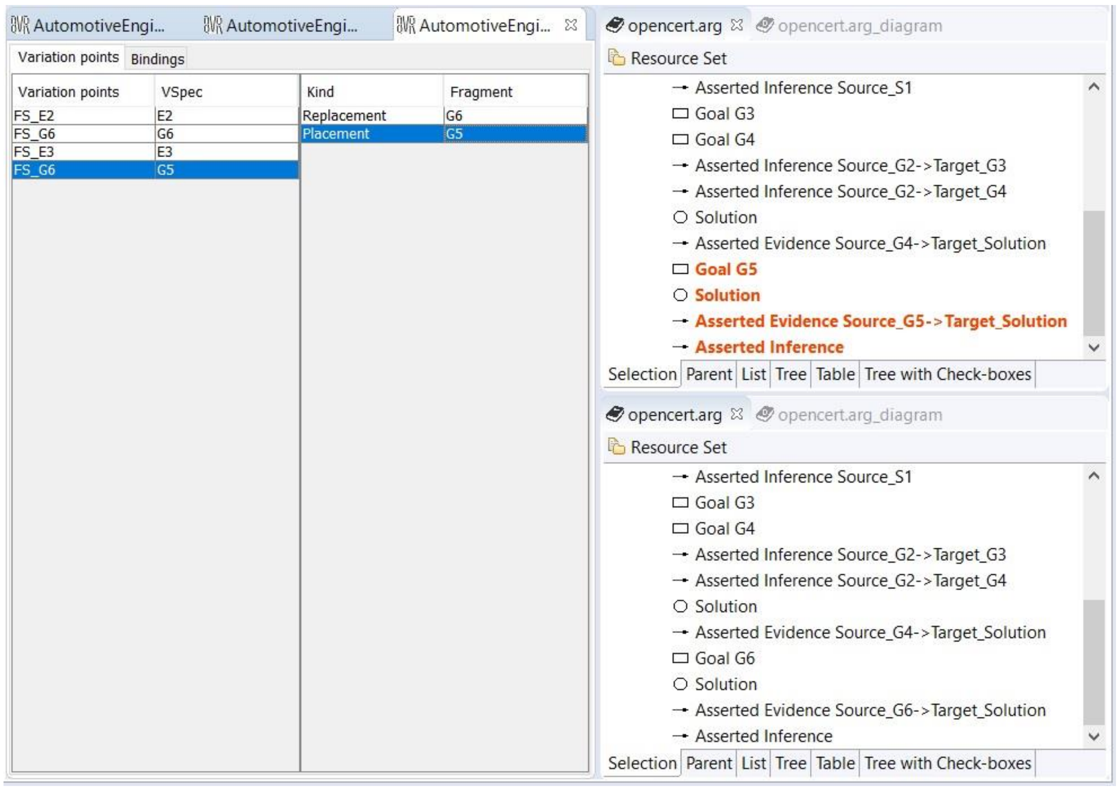
The Resolution models, shown in Figure 48, represents the two possible argumentation configurations (one in case of Truck and the other in case of a vehicle type different from Truck (Other)).



Valid: true []

Figure 48. Resolution models

The argumentation is stored in .arg and .arg_diagram files. The realization fragments have been specified for the .arg model. Figure 49 represents the Realization model. In this realization, the other configuration of the argumentation is realised. Based on the choices set within the resolution model, substitutions are specified and executed. Based on the specified substitutions, left part of Figure 49, some elements of a placement fragment are removed, and some elements of a replacement are injected.



Variation points	VSpec
FS_E2	E2
FS_G6	G6
FS_E3	E3
FS_G6	G5

Kind	Fragment
Replacement	G6
Placement	G5

opencert.arg · opencert.arg_diagram

Resource Set

- Asserted Inference Source_S1
- Goal G3
- Goal G4
- Asserted Inference Source_G2->Target_G3
- Asserted Inference Source_G2->Target_G4
- Solution
- Asserted Evidence Source_G4->Target_Solution
- **Goal G5**
- **Solution**
- **Asserted Evidence Source_G5->Target_Solution**
- **Asserted Inference**

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

opencert.arg · opencert.arg_diagram

Resource Set

- Asserted Inference Source_S1
- Goal G3
- Goal G4
- Asserted Inference Source_G2->Target_G3
- Asserted Inference Source_G2->Target_G4
- Solution
- Asserted Evidence Source_G4->Target_Solution
- Goal G6
- Solution
- Asserted Evidence Source_G6->Target_Solution
- Asserted Inference

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

Figure 49. Realization model

The execution of resolution generates the assurance case model. The re-configured model and diagram are updated in the assurance case editor of the OpenCert tool. The new configuration, where G6 is present instead of G5, is shown in Figure 50.

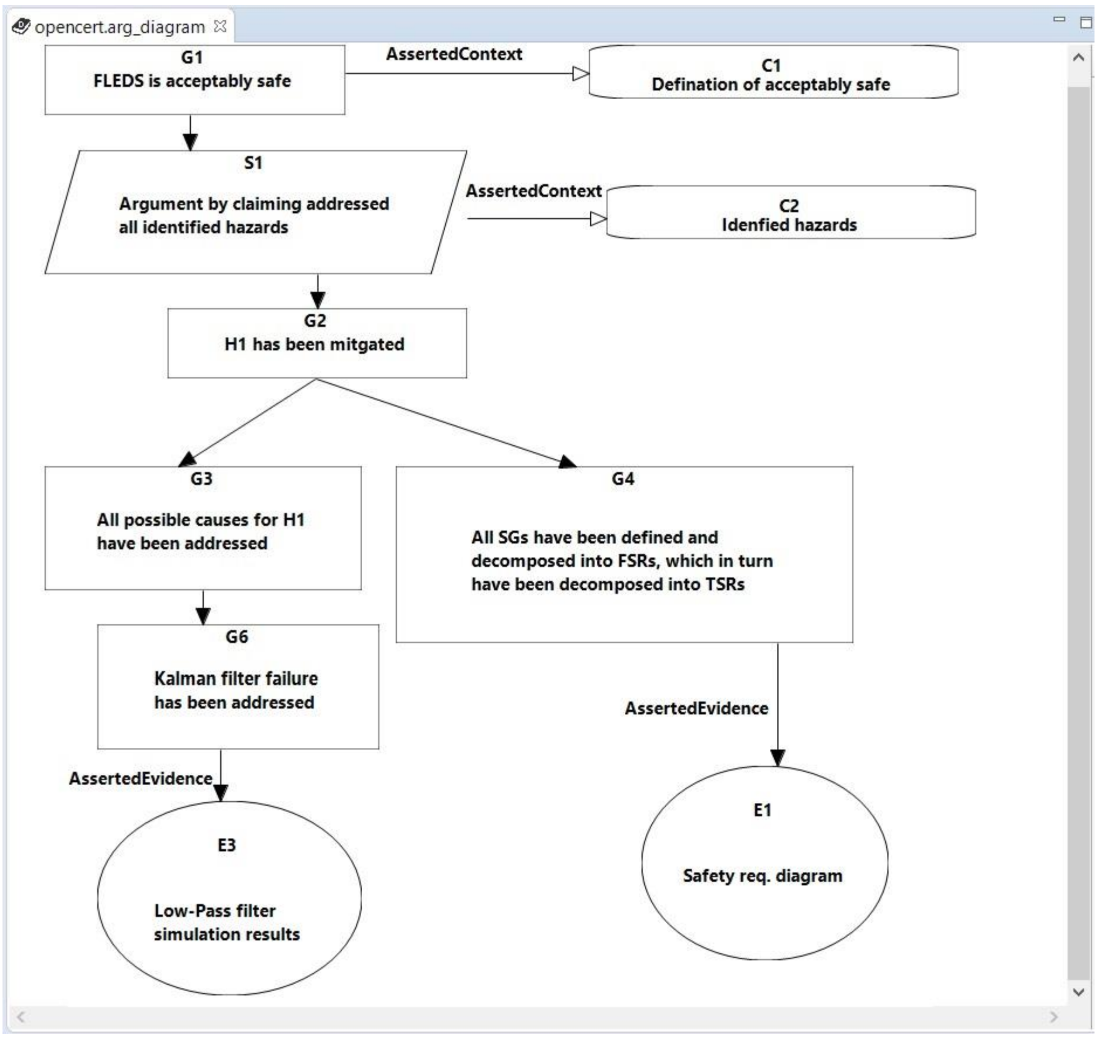

opencert.arg_diagram
G1
FLEDS is acceptably safe
AssertedContext
C1
Defination of acceptably safe
S1
Argument by claiming addressed all identified hazards
AssertedContext
C2
Idenfied hazards
G2
H1 has been mitgated
G3
All possible causes for H1 have been addressed
G4
All SGs have been defined and decomposed into FSRs, which in turn have been decomposed into TSRs
G6
Kalman filter failure has been addressed
AssertedEvidence
AssertedEvidence
E3
Low-Pass filter simulation results
E1
Safety req. diagram

Figure 50. Re-configured argumentation fragment

3.4 Automatic Generation of Arguments

This section provides the methodological guide for the automatic generation of process and product-based arguments within the AMASS Platform.

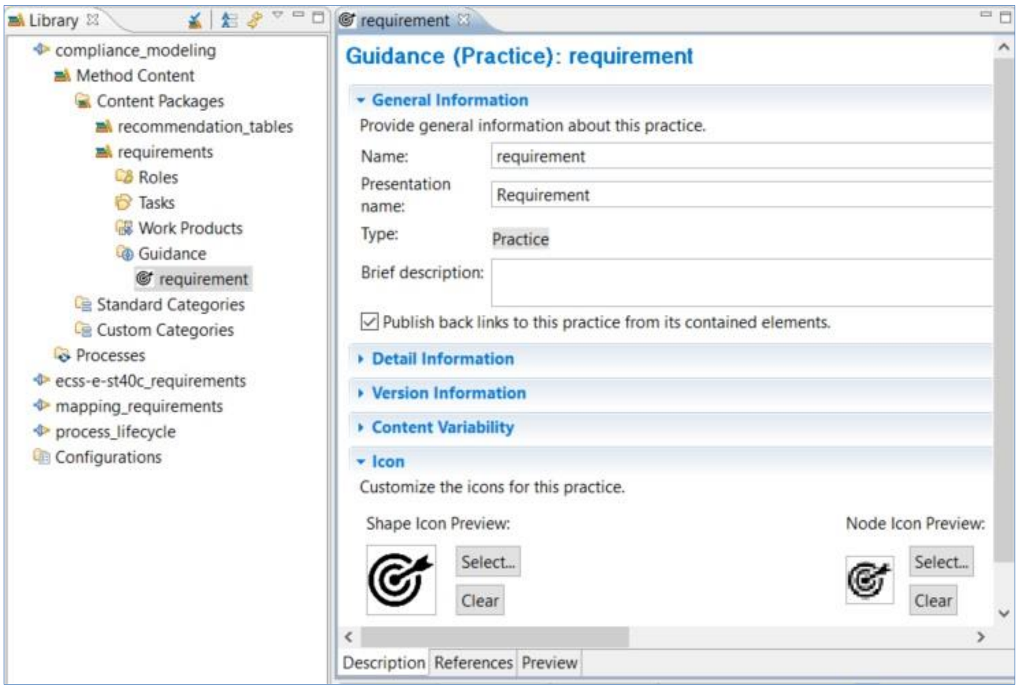
A safety case is a contextualized structured argument composed of process and product-based arguments to show that a system is acceptably safe. In order to reduce time and cost for the creation of safety arguments, support for automatic generation of safety arguments has been implemented [13]. The generated arguments are compliant to the CACM/SACM [30] metamodel and are rendered via a concrete syntax, e.g. GSN (Goal Structuring Notation).

3.4.1 Automatic generation of process-based argumentation representing plans (*)

3.4.1.1 Modelling of standards requirements and safety processes (*)

As shown in Figure 15, the first step involves modelling of standard's requirements and safety process in EPF-C according to the best practices as well as according to the standard(s) such as ECSS-E-ST-40C [27]. In this section, AOCS, will be described to illustrate how this functionality works to detect omission of key evidence fallacy and generates the process-based argumentation. AOCS software development process follows a set of recommendations from the ECSS-E-ST-40C. The ECSS-E-ST-40C standard explicitly describes the requirements and recommendations related to planning process such as activities, tasks and expected outputs. However, requirements related to the staffing plan (i.e., key competencies or skills required for specific roles) and tool qualifications are not provided. Key competencies required for an architect/designer role are adapted from railway standard EN 50128 [21]. The key competencies for other roles are adapted from industrial requirements, specifically for AOCS.

Similarly, the requirements related to tool qualifications are adapted. The requirements and process are modelled as plugins in EPF-C by following the guidelines mentioned by IBM approach [49]. For this reason, IBM recommended to create three separate plugins, in particular, capturing standard requirements, modelling process lifecycle (i.e., content elements and process) and mapping standard requirements. However, EPF-C does not support the definition of a user-defined type. Therefore, a new Practice named “requirement” has been defined in the compliance_modeling plugin, and a new icon has been included in the icon section of the “Description” tab (see Figure 51). In the ecss-e-st40c_requirements plugin, the guidance type “Practice” has been customized with an icon and variability relationship “Extends” with compliance_modeling plugin. Figure 52 shows the requirement modelling of an AOCS Engineer. It is assumed that the user reads the standard and identifies the keywords such as system, hazards, and software etc. These keywords should be included in both requirements and process elements. In case there are more than one requirement that needs to be fulfilled, a semicolon (;) is used to separate them in the “Brief description” field. The requirements can also be automatically imported into the EPF-C. The instructions for converting an excel file to XML format compatible with EPF-C has been described in AMASS D1.5 [1].

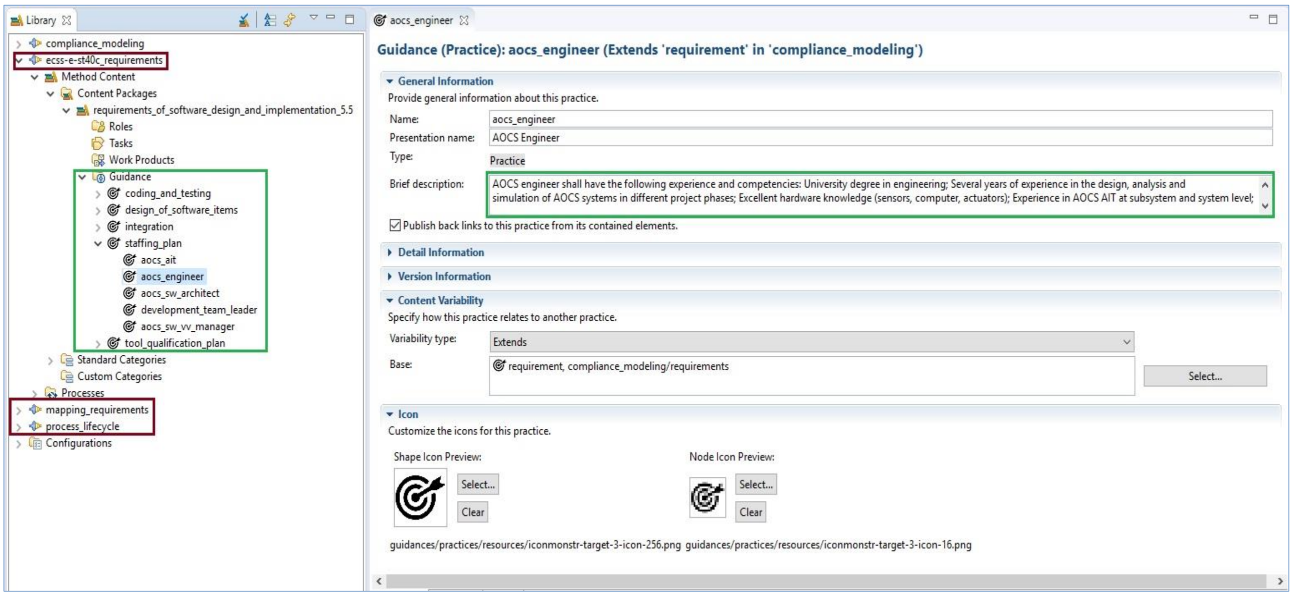

Library
compliance_modeling
Method Content
Content Packages
recommendation_tables
requirements
Roles
Tasks
Work Products
Guidance
requirement
Standard Categories
Custom Categories
Processes
ecss-e-st40c_requirements
mapping_requirements
process_lifecycle
Configurations

requirement

Guidance (Practice): requirement

General Information
Provide general information about this practice.
Name: requirement
Presentation name: Requirement
Type: Practice
Brief description:
☑ Publish back links to this practice from its contained elements.
Detail Information
Version Information
Content Variability
Icon
Customize the icons for this practice.
Shape Icon Preview:
Node Icon Preview:

Figure 51. Customization of requirement practice



Guidance (Practice): aocs_engineer (Extends 'requirement' in 'compliance_modeling')

General Information
Provide general information about this practice.

Name: aocs_engineer
Presentation name: AOCS Engineer
Type: Practice
Brief description: AOCS engineer shall have the following experience and competencies: University degree in engineering; Several years of experience in the design, analysis and simulation of AOCS systems in different project phases; Excellent hardware knowledge (sensors, computer, actuators); Experience in AOCS AIT at subsystem and system level;

☑ Publish back links to this practice from its contained elements.

Content Variability
Specify how this practice relates to another practice.

Variability type: Extends
Base: requirement, compliance_modeling/requirements

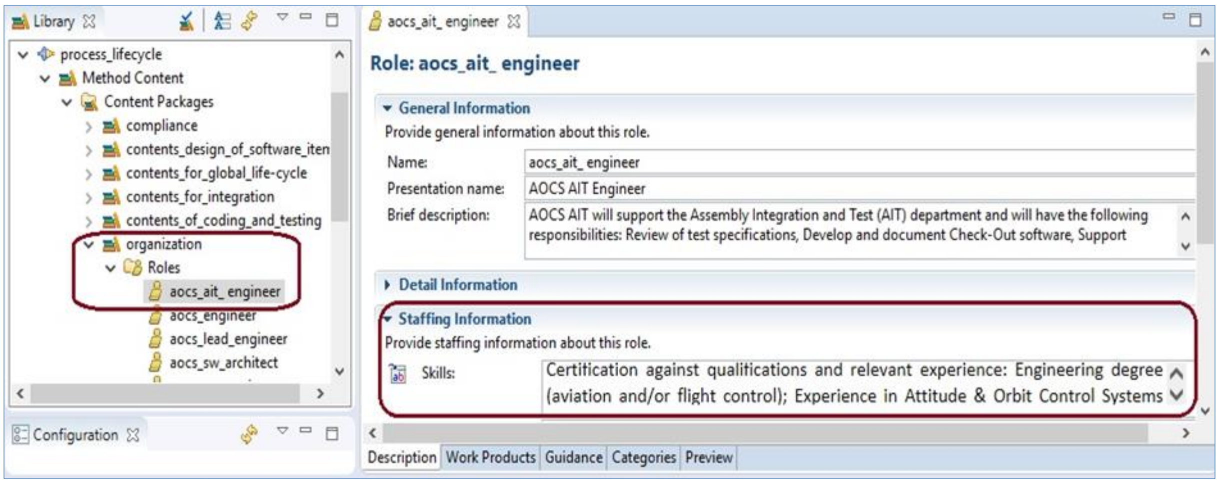
Icon
Customize the icons for this practice.

Shape Icon Preview: guidances/practices/resources/iconmonstr-target-3-icon-256.png
Node Icon Preview: guidances/practices/resources/iconmonstr-target-3-icon-16.png

Figure 52. Modelling ECSS-E-ST-40C standard requirements for AOCS

The guidance for the creation of process elements is available in the EPF-C manual [1]. The properties/fields in EPF-C used to model the evidences related to process elements are as follows:

- *Skills* (Staffing Information) field in EPF-C is used to model the qualifications provided by instances of a role who has the responsibilities to perform relevant tasks correctly and efficiently. If a role has more than one requirement to fulfil, a semicolon (;) should separate the presented evidences or rationales that correspond to different requirements. Figure 53 shows the modelling of key competencies of AOCS AIT (Assembly Integration and Test) Engineer.
- *Key considerations* (Detail Information) field of Tools is used to specify certifications or rationales against required tool qualifications (see Figure 54). As mentioned above, the same keywords should be included in both requirements and process elements.



Role: aocs_ait_ engineer

General Information
Provide general information about this role.

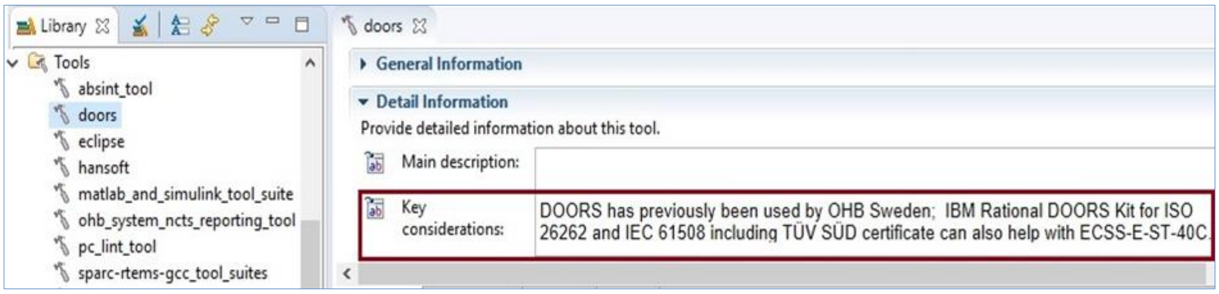
Name: aocs_ait_ engineer
Presentation name: AOCS AIT Engineer
Brief description: AOCS AIT will support the Assembly Integration and Test (AIT) department and will have the following responsibilities: Review of test specifications, Develop and document Check-Out software, Support

Staffing Information
Provide staffing information about this role.

Skills: Certitication against qualifications and relevant experience: Engineering degree (aviation and/or flight control); Experience in Attitude & Orbit Control Systems

Figure 53. Modelling evidence for AOCS AIT Engineer role in process lifecycle plugin

In order to define the mapping, standard requirements are copied in the *mapping_requirements* plugin. These copied requirements have a variability relationship “Contributes” with original requirements. In addition, the links between requirements and lifecycle elements have been established through “References”, as shown in Figure 55.

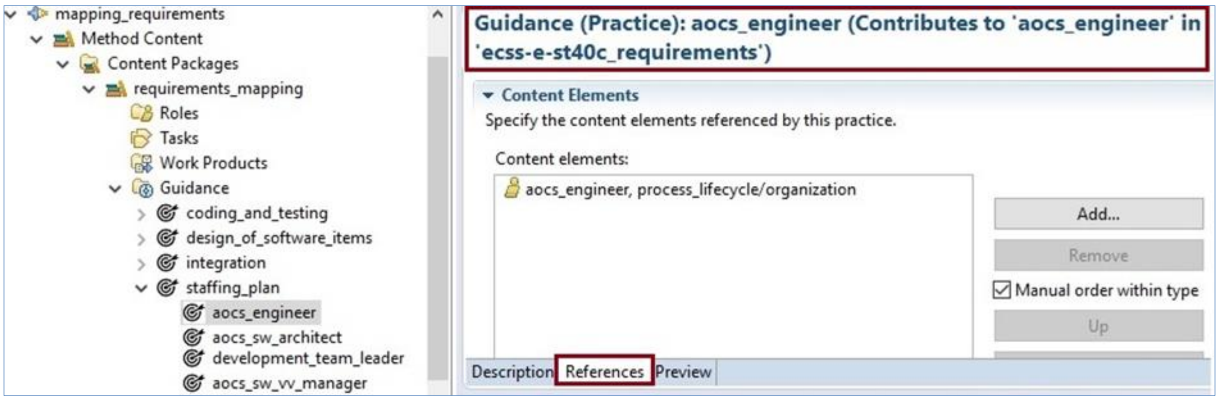


Provide detailed information about this tool.

Main description:

Key considerations: DOORS has previously been used by OHB Sweden; IBM Rational DOORS Kit for ISO 26262 and IEC 61508 including TÜV SÜD certificate can also help with ECSS-E-ST-40C.

Figure 54. Modelling evidence for Tool Qualification in process lifecycle plugin



Guidance (Practice): aocs_engineer (Contributes to 'aocs_engineer' in 'ecss-e-st40c_requirements')

Content Elements

Specify the content elements referenced by this practice.

Content elements:

aocs_engineer, process_lifecycle/organization

Figure 55. Mapping standard requirements

3.4.1.2 Detecting fallacies in process models (*)

Within the context of fallacy free process-based arguments, safety processes mandated by prescriptive standards such as ECSS-E-ST-40C [27] are considered, also the process related structures in UMA metamodel (managed by EPF-C) that represent plans have been identified. For instance, a basic process structure consists of tasks, and a set of work products, roles, tools and guidance (such as checklists, tool mentors, guidelines, and examples).

In order to prevent or detect fallacies in the process-based argumentations, a *fallacy detection* plugin has been implemented that validates whether the safety process contains the sufficient information corresponding to the key evidence for supporting the specific requirement. In particular, the plugin prevents the occurrence of fallacy, specifically, “*Omission of Key Evidence”* in process-based arguments. If no or less evidences are provided to support the claim (i.e., at least one of the evidence has been omitted) or no valid reasons (rationales) are given for its omission; it means that the process contains the omission of key evidence fallacies [50].The *fallacy detection* plugin is invoked from a right-click menu on those elements of type *ProcessComponent* (*Capability Pattern* or *Delivery Process*) since these elements have all the information of the modelled process, specifically select the “Fallacy Detection → Detect Omission of Key Evidence” option, as shown in Figure 56. After invoking the plugin, a pop-up dialogue appears to ask the user where he or she wants to store the results of the validation (see Figure 57).

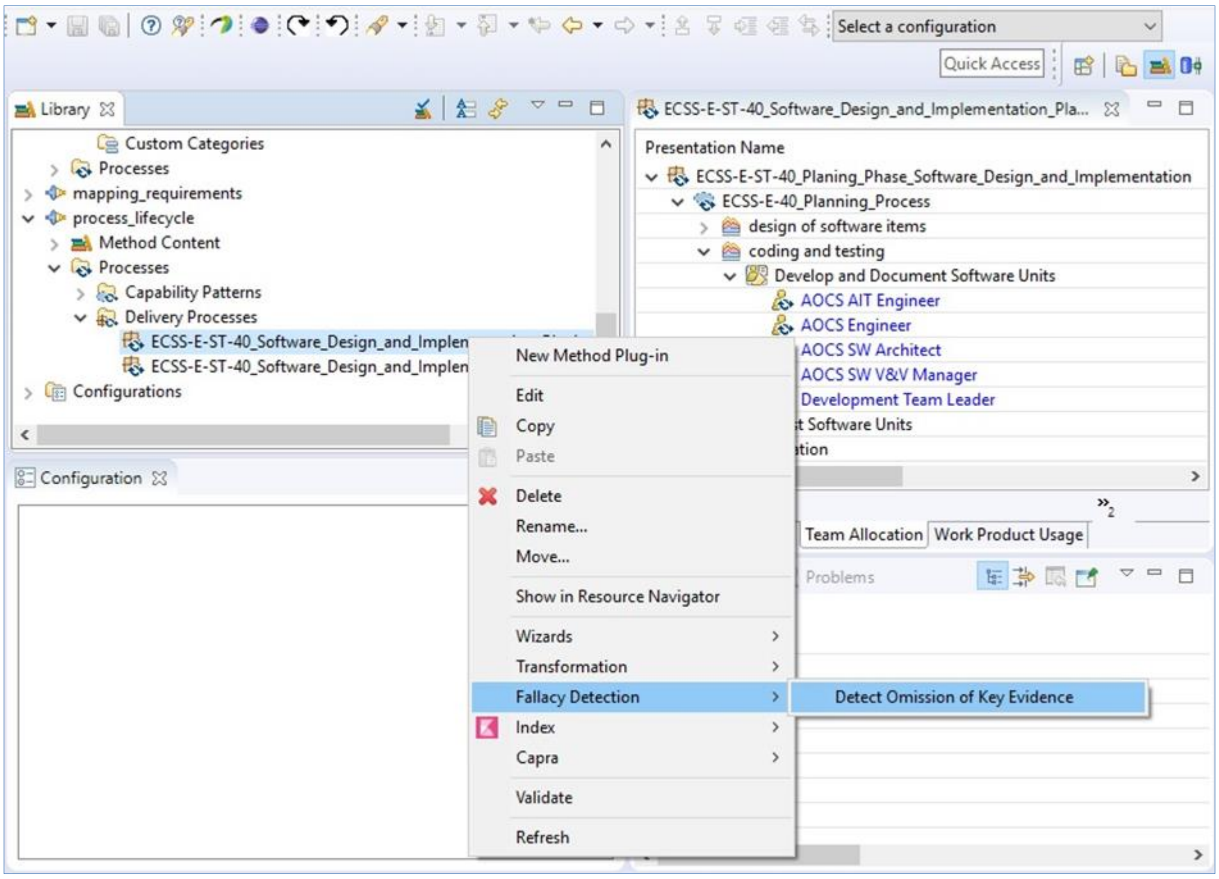
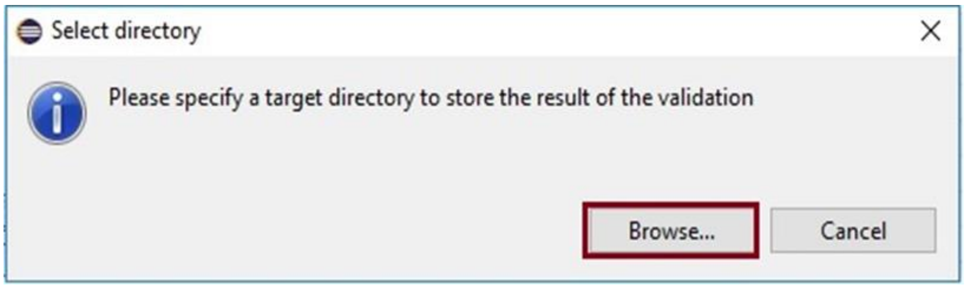


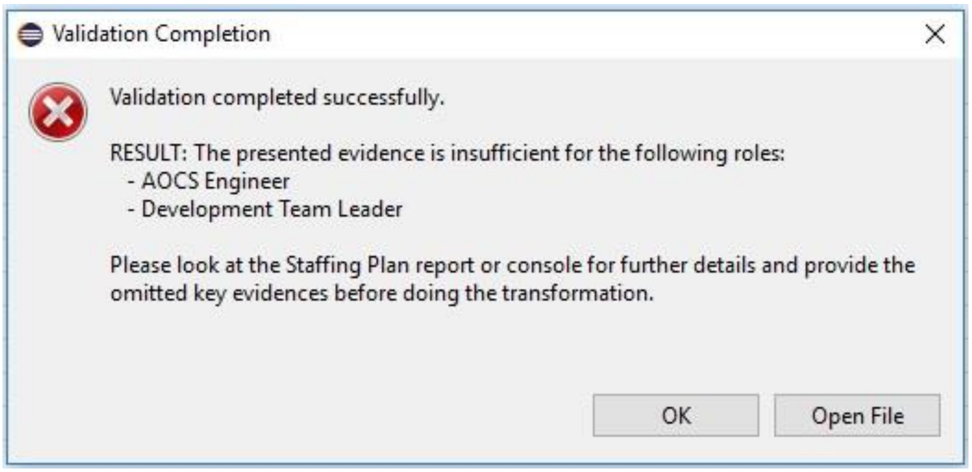
Figure 56. Initiating the fallacy detection plugin


Select directory

Please specify a target directory to store the result of the validation

Figure 57. Selecting the target directory for storing results

After selecting the target directory, the validation process starts. When the fallacies detection process has finished, the window appears (see Figure 58) for allowing user to open the validation report in the form of TXT file by clicking the button “Open File”. In case of more than one validation reports, the “Open Folder” message appears that allows the user to open the target folder by clicking the “Open Folder” button, as shown in Figure 59. The validation results are also printed on the console (see Figure 60).



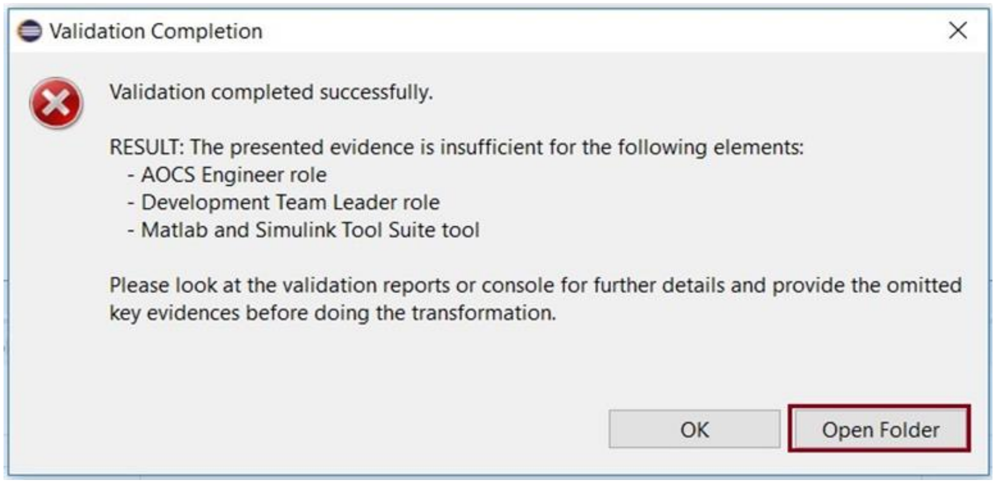
Validation Completion

Validation completed successfully.

RESULT: The presented evidence is insufficient for the following roles:
- AOCS Engineer
- Development Team Leader

Please look at the Staffing Plan report or console for further details and provide the omitted key evidences before doing the transformation.

Figure 58. Opening validation result file



Validation Completion

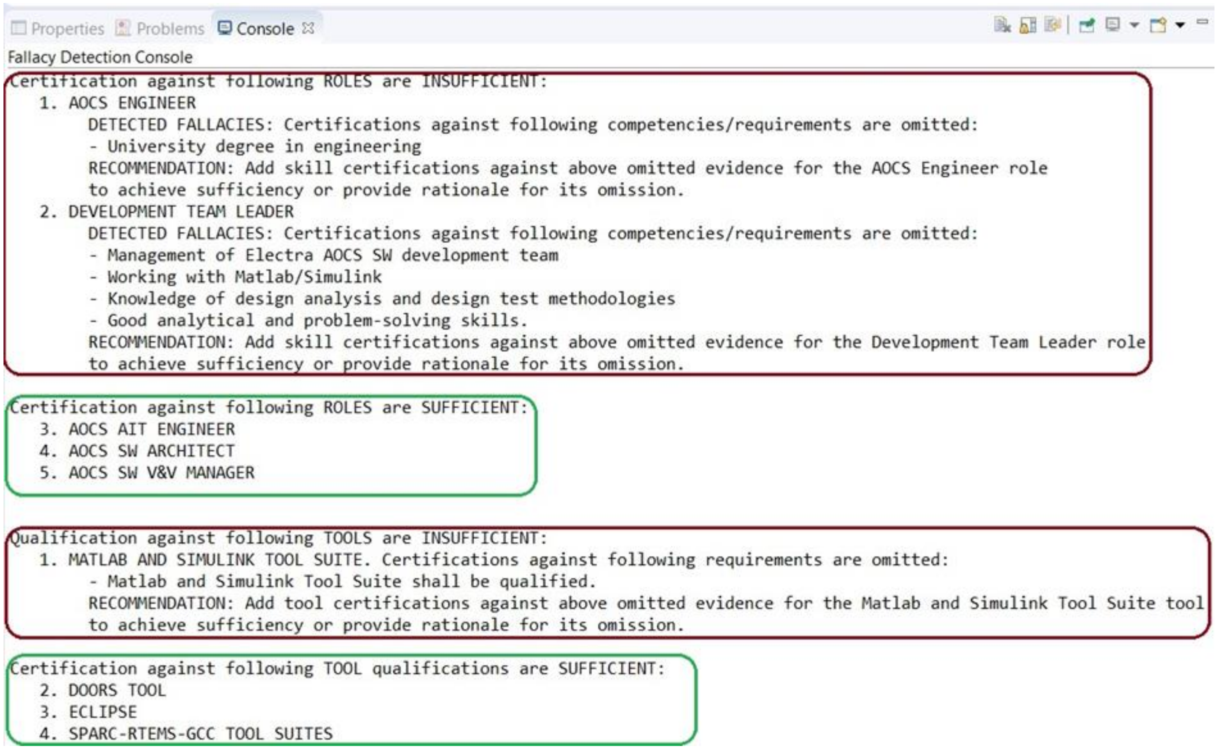
Validation completed successfully.

RESULT: The presented evidence is insufficient for the following elements:
- AOCS Engineer role
- Development Team Leader role
- Matlab and Simulink Tool Suite tool

Please look at the validation reports or console for further details and provide the omitted key evidences before doing the transformation.

Figure 59. Opening validation result folder

Figure 60 shows the results including the list of roles and tools containing insufficient information (i.e., detected fallacies) and recommendations that are enclosed in the red box, and the elements containing sufficient evidences (enclosed in the green box). By looking at results, it is found that skill certifications (evidences) against AOCS AIT engineer, AOCS SW (SoftWare) Architect and AOCS SW V&V (Verification and Validation) Manager are sufficient, and the presented tool qualifications for the *Tools* DOORS, Eclipse, and SPARC-RTEMS-GCC are sufficient. However, Key evidences associated to AOCS Engineer and Development Team Leader *Roles*, and the Matlab and Simulink Tool Suite *Tool* are omitted. In addition, no valid reasoning (rationale) is given for the omissions. Based on the results, engineers can either modify the process models or provide the rationale about omitted information by following the recommendations.

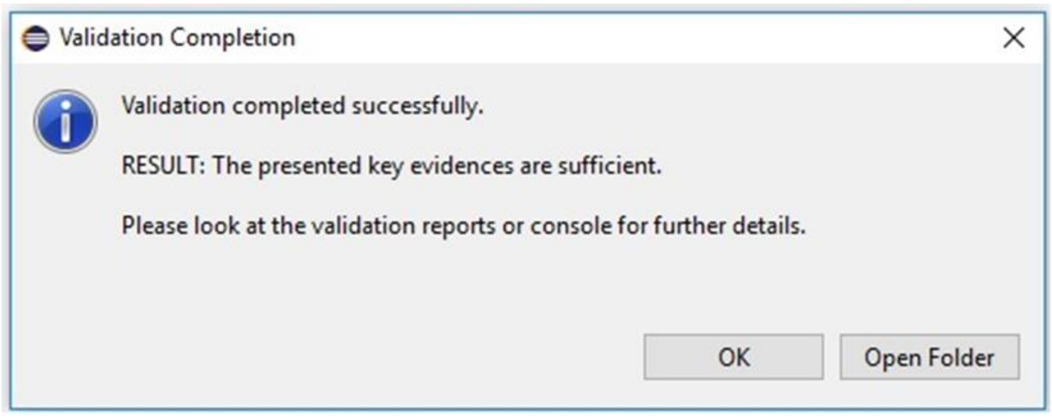

Properties Problems Console

Fallacy Detection Console

```
Certification against following ROLES are INSUFFICIENT:
  1. AOCS ENGINEER
       DETECTED FALLACIES: Certifications against following competencies/requirements are omitted:
       - University degree in engineering
       RECOMMENDATION: Add skill certifications against above omitted evidence for the AOCS Engineer role
       to achieve sufficiency or provide rationale for its omission.
  2. DEVELOPMENT TEAM LEADER
       DETECTED FALLACIES: Certifications against following competencies/requirements are omitted:
       - Management of Electra AOCS SW development team
       - Working with Matlab/Simulink
       - Knowledge of design analysis and design test methodologies
       - Good analytical and problem-solving skills.
       RECOMMENDATION: Add skill certifications against above omitted evidence for the Development Team Leader role
       to achieve sufficiency or provide rationale for its omission.

Certification against following ROLES are SUFFICIENT:
  3. AOCS AIT ENGINEER
  4. AOCS SW ARCHITECT
  5. AOCS SW V&V MANAGER

Qualification against following TOOLS are INSUFFICIENT:
  1. MATLAB AND SIMULINK TOOL SUITE. Certifications against following requirements are omitted:
       - Matlab and Simulink Tool Suite shall be qualified.
       RECOMMENDATION: Add tool certifications against above omitted evidence for the Matlab and Simulink Tool Suite tool
       to achieve sufficiency or provide rationale for its omission.

Certification against following TOOL qualifications are SUFFICIENT:
  2. DOORS TOOL
  3. ECLIPSE
  4. SPARC-RTEMS-GCC TOOL SUITES
```

Figure 60. Printed results on the console

Once the fallacies are detected, they are eliminated by modifying the process models and re-running the validation process yielded no further fallacies. Figure 61 shows the dialogue shown once all the previous evidences have been added to the corresponding elements and all the requirements have been fully satisfied. Then the fallacy free process-based argumentations will be automatically generated from the modified models.


Validation Completion

Validation completed successfully.

RESULT: The presented key evidences are sufficient.

Please look at the validation reports or console for further details.

Figure 61. Selecting target directory for storing results

3.4.1.3 Generating process-based argumentation (*)

Process-based argumentations are automatically generated from the modified process models by applying model-to-model transformation (given in Epsilon [42]), implemented as *Argument Generator* plugin. The generated arguments are visualized via the Assurance editor in the OpenCert tool, as shown in Figure 62. The generated arguments are compliant to the CACM/SACM [30] metamodel and are rendered via a concrete syntax, e.g. GSN (Goal Structuring Notation).

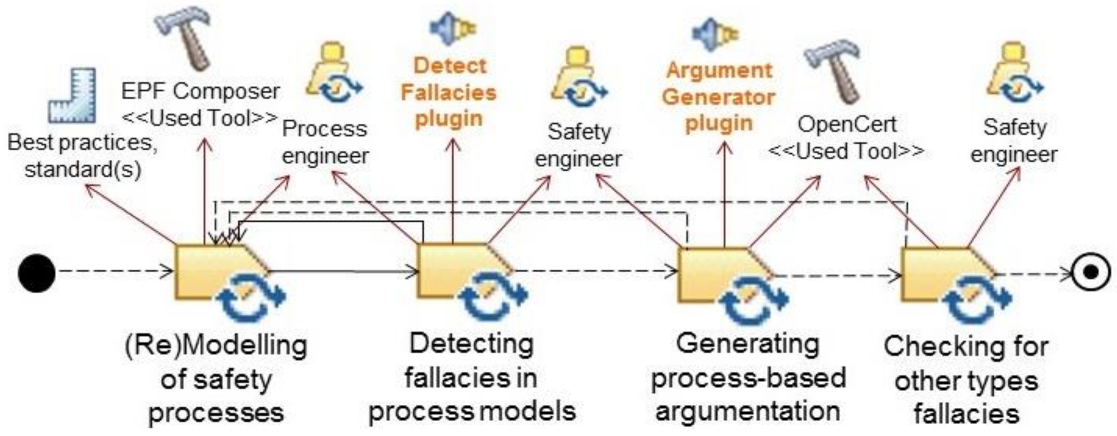
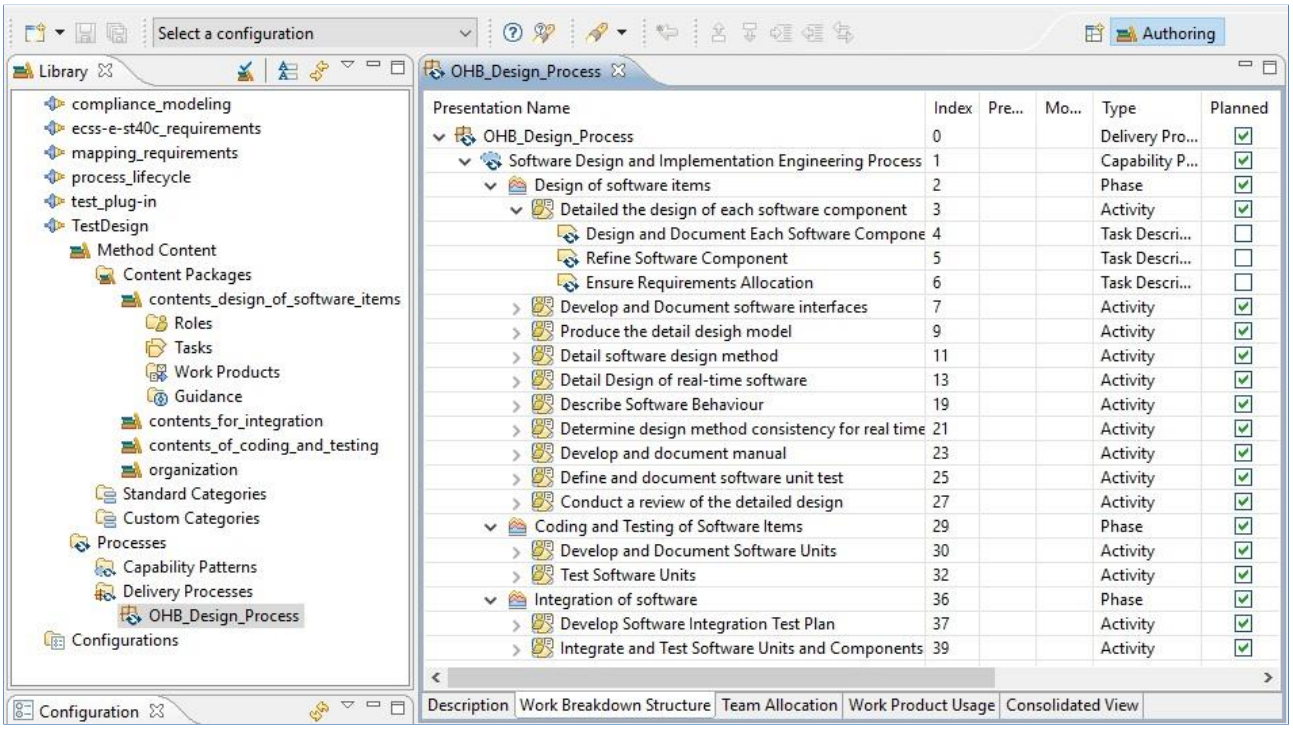

EPF Composer
<<Used Tool>>
Best practices, standard(s)
Process engineer
Detect Fallacies plugin
Safety engineer
Argument Generator plugin
OpenCert
<<Used Tool>>
Safety engineer
(Re)Modelling of safety processes
Detecting fallacies in process models
Generating process-based argumentation
Checking for other types fallacies

Figure 62. Process-based arguments generation

Within the context of process-based arguments, safety processes mandated by prescriptive standards such as ECSS-E-ST-40C [27] are considered, also the process related structures in UMA metamodel (managed by EPF-C) that represent plans have been identified (see Figure 63). It might be noted that the EPF-C [25] is based on the UMA metamodel, which is an evolution of the OMG's SPEM 1.1 [28]. The major parts of UMA are incorporated in SPEM 2.0 [29]. For the generation of process-based arguments, the *Work Breakdown Structure* of processes is considered. The "ProcessComponent" that contains the information of the process is mapped into a "Case", whereas the planning phases of ECSS-E-ST-40C (e.g., design of software items, coding and testing of software items, integration of software) are constituted of the top-level claims (aka GSN Goals) stating that the planned process is in compliance with the required standard-level.

These claims are decomposed into the process activities (e.g., Test Software Unit) for showing that all the activities have been planned, in turn, for each activity all the tasks have been planned (e.g., Develop and Document Test Procedure) and so on. The crucial process elements that are associated to a task, namely a set of roles, a set of work products, and a set of guidance are mapped into the sub-claims. The *Purpose* field of the *Delivery Process* element in the EPF-C is used to model the concept of *Context* which will be attached to the claim corresponding to a phase. *ArgumentReasoning* (aka GSN strategy) is created in order to divide the claim into sub-claims. The evidences related to these elements are mapped into the InformationElementCitation property type "solutions". The relationship AssertedInference (aka GSN *SolvedBy*) is created to link claim to strategy or claim to claim, i.e., a target which is not a solution. The relationship AssertedEvidence (aka GSN *SolvedBy*) is created to link a claim to a solution. The relationship Asserted (aka GSN *inContextOf*) is created to link a claim to a context, claim to assumption, and claim to justification. This mapping is coherent with what was initially conceived in context of the SafeCer project and published in [34]. The rules that apply for the transformation are described in AMASS D6.3 [12], also published in [50].

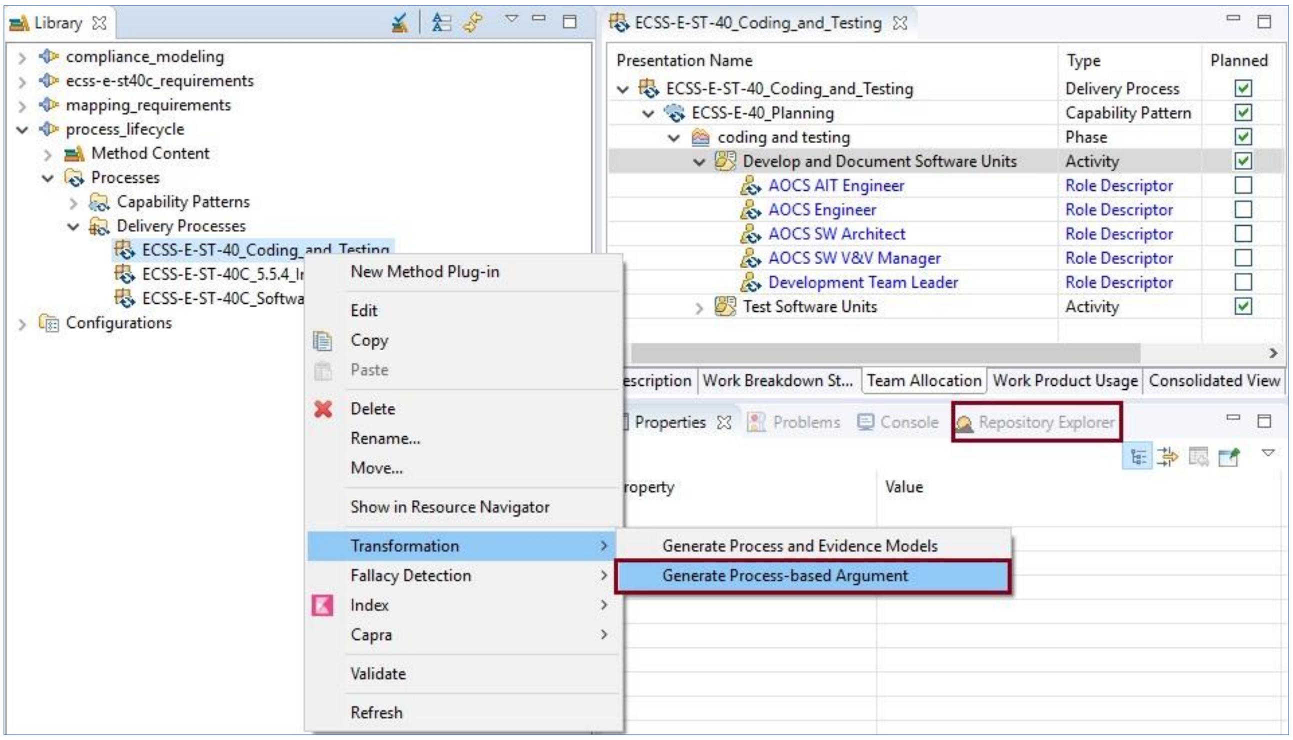


- compliance_modeling
- ecss-e-st40c_requirements
- mapping_requirements
- process_lifecycle
- test_plug-in
- TestDesign
  - Method Content
    - Content Packages
      - contents_design_of_software_items
        - Roles
        - Tasks
        - Work Products
        - Guidance
      - contents_for_integration
      - contents_of_coding_and_testing
      - organization
    - Standard Categories
    - Custom Categories
  - Processes
    - Capability Patterns
    - Delivery Processes
      - OHB_Design_Process
- Configurations

Presentation Name	Index	Pre...	Mo...	Type	Planned
OHB_Design_Process	0			Delivery Pro...	✓
Software Design and Implementation Engineering Process	1			Capability P...	✓
Design of software items	2			Phase	✓
Detailed the design of each software component	3			Activity	✓
Design and Document Each Software Compone	4			Task Descri...	☐
Refine Software Component	5			Task Descri...	☐
Ensure Requirements Allocation	6			Task Descri...	☐
Develop and Document software interfaces	7			Activity	✓
Produce the detail desigh model	9			Activity	✓
Detail software design method	11			Activity	✓
Detail Design of real-time software	13			Activity	✓
Describe Software Behaviour	19			Activity	✓
Determine design method consistency for real time	21			Activity	✓
Develop and document manual	23			Activity	✓
Define and document software unit test	25			Activity	✓
Conduct a review of the detailed design	27			Activity	✓
Coding and Testing of Software Items	29			Phase	✓
Develop and Document Software Units	30			Activity	✓
Test Software Units	32			Activity	✓
Integration of software	36			Phase	✓
Develop Software Integration Test Plan	37			Activity	✓
Integrate and Test Software Units and Components	39			Activity	✓

Figure 63. EPF-C: ECSS process fragment

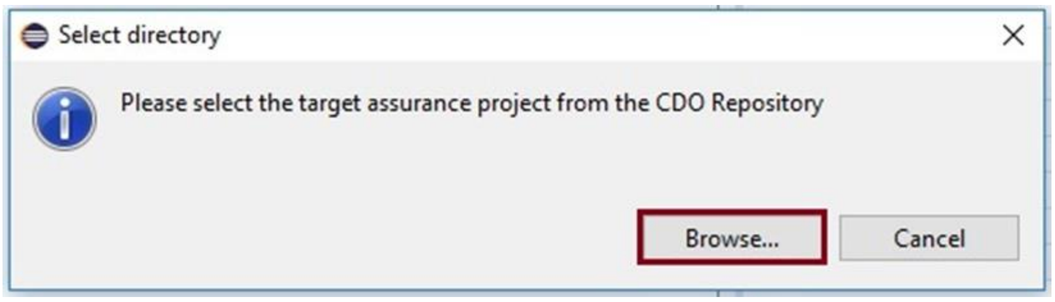
To generate process-based arguments automatically, a plugin has been implemented in AMASS D6.6 [13]. The overall workflow for the automatic generation of process-based arguments is presented in Figure 15. In this context, the argumentation generation is invoked from a right-click menu of *ProcessComponent* (*Capability Pattern* or *Delivery* Process) modelled in EPF-C, in particular, by selecting the "Transformation → Generate Process-based Argument" option, as shown in Figure 64.



- compliance_modeling
- ecss-e-st40c_requirements
- mapping_requirements
- process_lifecycle
  - Method Content
  - Processes
    - Capability Patterns
    - Delivery Processes
      - ECSS-E-ST-40_Coding_and_Testing
      - ECSS-E-ST-40C_5.5.4_I
      - ECSS-E-ST-40C_Softwa
- Configurations

Presentation Name	Type	Planned
ECSS-E-ST-40_Coding_and_Testing	Delivery Process	✓
ECSS-E-40_Planning	Capability Pattern	✓
coding and testing	Phase	✓
Develop and Document Software Units	Activity	✓
AOCS AIT Engineer	Role Descriptor	☐
AOCS Engineer	Role Descriptor	☐
AOCS SW Architect	Role Descriptor	☐
AOCS SW V&V Manager	Role Descriptor	☐
Development Team Leader	Role Descriptor	☐
Test Software Units	Activity	✓

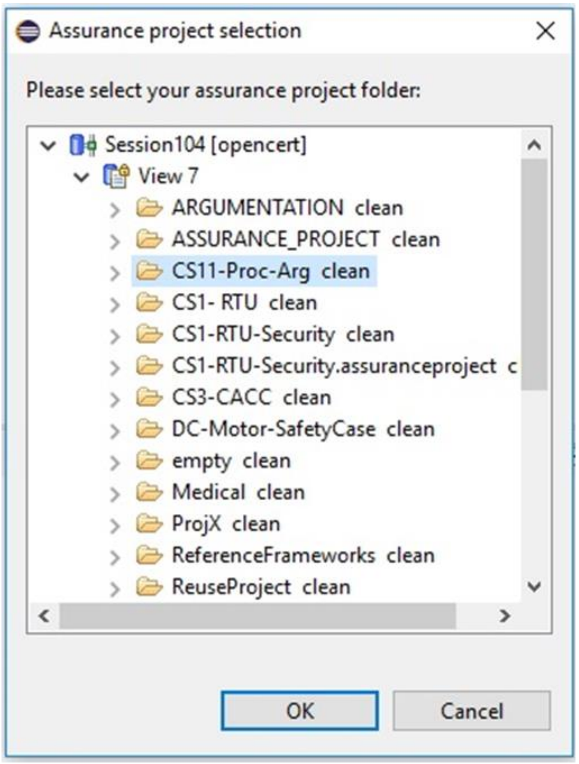
Figure 64. Initiating the process-based argumentation generator plugin


Select directory

Please select the target assurance project from the CDO Repository

Figure 65. Browse button for CDO Repository

Once the transformation plugin is invoked, the dialogue box is displayed, in order to select the target *Assurance Project* from the CDO Repository using the "Browse" functionality (see Figure 65). Then, select a previously created *Assurance Project* folder to store the results of the transformation and press the "OK" button (see Figure 66). To do that it is necessary that the CDO Repository has already configured and the Repository Explorer has been opened. To configure the Repository, click "Window → Preferences". Then, in the Preferences window, open the "OpenCert" menu, click on "Model Repository Preferences", and fill the fields. To create an Assurance Project, click on "File → New → Other", select "New Assurance Project" and follow the instructions.

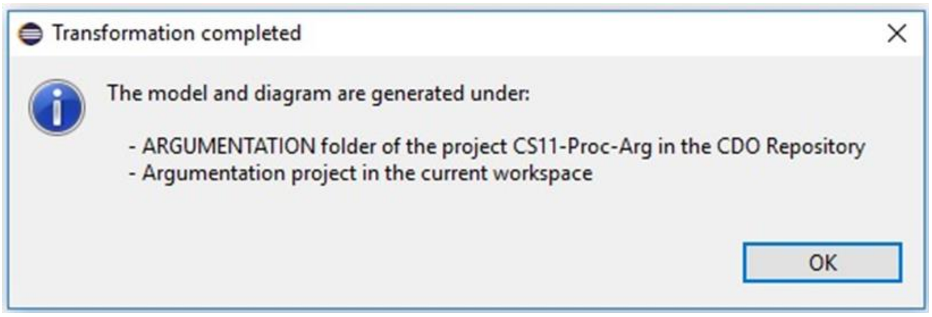

Assurance project selection

Please select your assurance project folder:

```
Session104 [opencert]
  View 7
    ARGUMENTATION  clean
    ASSURANCE_PROJECT  clean
    CS11-Proc-Arg  clean
    CS1- RTU  clean
    CS1-RTU-Security  clean
    CS1-RTU-Security.assuranceproject  c
    CS3-CACC  clean
    DC-Motor-SafetyCase  clean
    empty  clean
    Medical  clean
    ProjX  clean
    ReferenceFrameworks  clean
    ReuseProject  clean
```

Figure 66. Selecting an Assurance Project

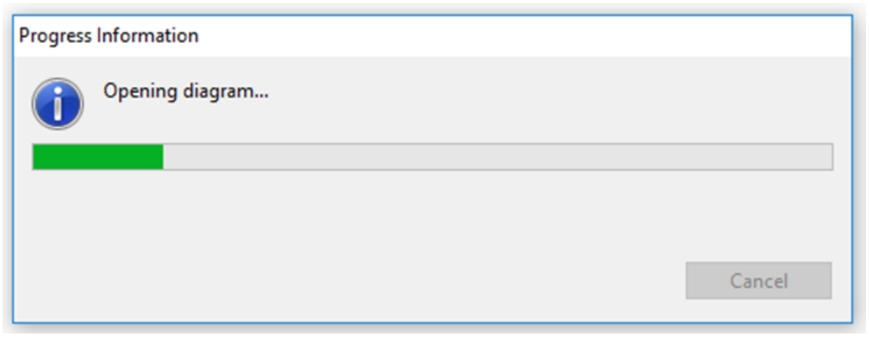
After selecting the Assurance Project folder, the transformation process starts. Then, a dialogue window appears to inform the user that the transformation process has finished successfully and listing the locations where the generated files have been stored (see Figure 67). By clicking on the "OK" button, the progress information monitor is displayed to inform the user that the diagram is opening, as shown in Figure 68.


Transformation completed

The model and diagram are generated under:

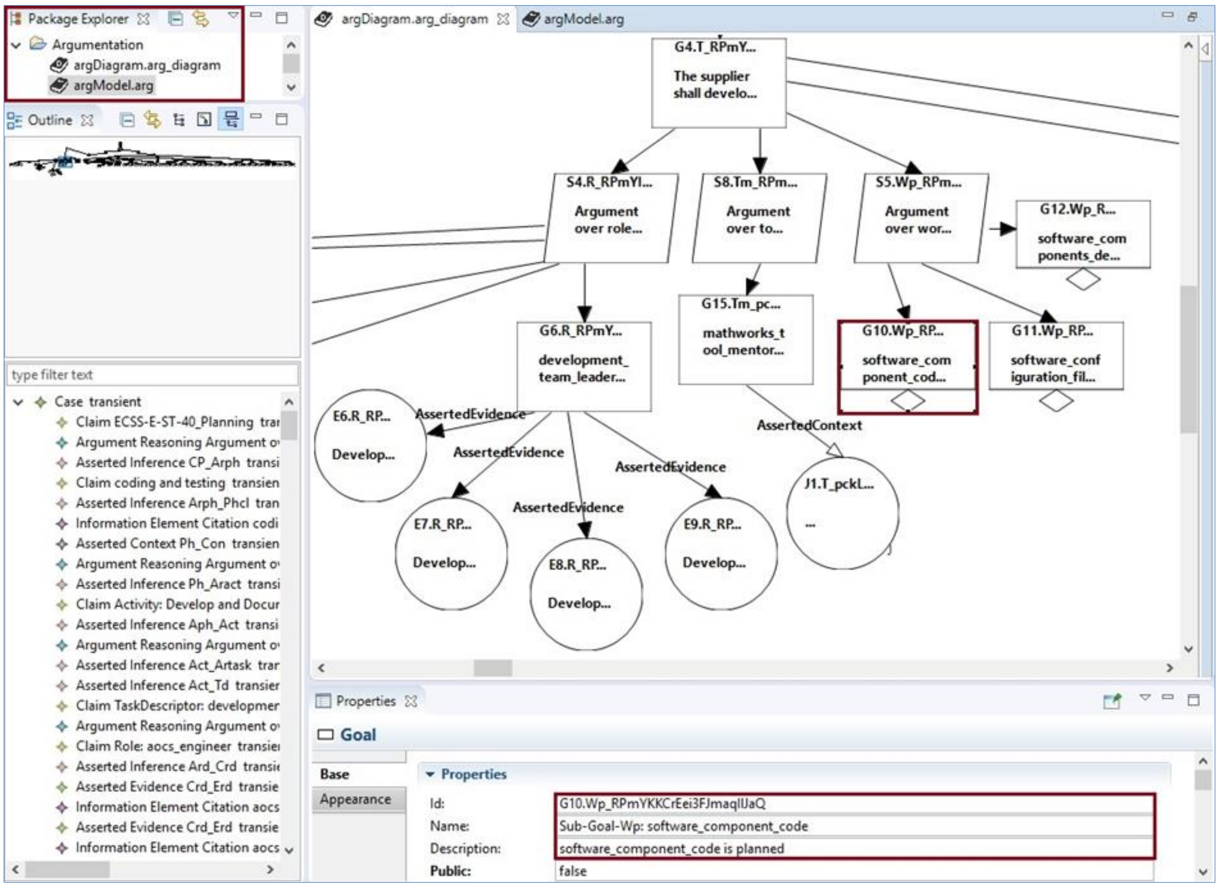
- ARGUMENTATION folder of the project CS11-Proc-Arg in the CDO Repository
- Argumentation project in the current workspace

Figure 67. Transformation successfully completed


Progress Information

Opening diagram...

Figure 68. Opening diagram

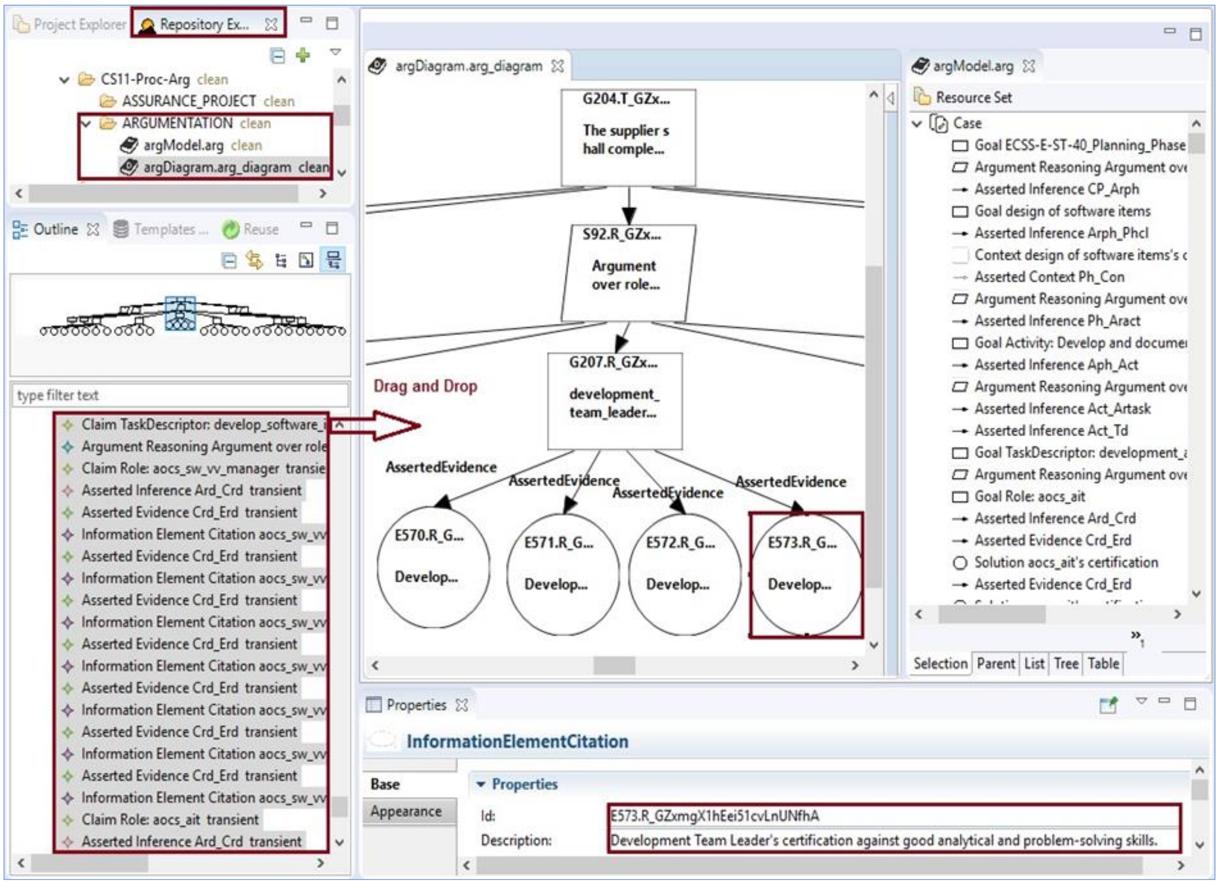

Package Explorer
Argumentation
argDiagram.arg_diagram
argModel.arg
Outline
type filter text
Case transient
Claim ECSS-E-ST-40_Planning trai
Argument Reasoning Argument o
Asserted Inference CP_Arph transi
Claim coding and testing transien
Asserted Inference Arph_Phcl tran
Information Element Citation codi
Asserted Context Ph_Con transien
Argument Reasoning Argument o
Asserted Inference Ph_Aract transi
Claim Activity: Develop and Docur
Asserted Inference Aph_Act transi
Argument Reasoning Argument o
Asserted Inference Act_Artask trar
Asserted Inference Act_Td transier
Claim TaskDescriptor: developmer
Argument Reasoning Argument o
Claim Role: aocs_engineer transie
Asserted Inference Ard_Crd transie
Asserted Evidence Crd_Erd transie
Information Element Citation aocs
Asserted Evidence Crd_Erd transie
Information Element Citation aocs
argDiagram.arg_diagram
argModel.arg
G4.T_RPmY... The supplier shall develo...
S4.R_RPmYI... Argument over role...
S8.Tm_RPm... Argument over to...
S5.Wp_RPm... Argument over wor...
G12.Wp_R... software_com ponents_de...
G15.Tm_pc... mathworks_t ool_mentor...
G6.R_RPmY... development_ team_leader...
G10.Wp_RP... software_com ponent_cod...
G11.Wp_RP... software_conf iguration_fil...
E6.R_RP... Develop...
E7.R_RP... Develop...
E8.R_RP... Develop...
E9.R_RP... Develop...
J1.T_pckl... ...
AssertedEvidence
AssertedEvidence
AssertedEvidence
AssertedEvidence
AssertedContext
Properties
Goal
Base
Appearance
Properties

Id: G10.Wp_RPmYKKCrEei3FJmaqlUaQ
Name: Sub-Goal-Wp: software_component_code
Description: software_component_code is planned
Public: false

Figure 69. Locally generated argumentation model and diagram

Once the transformation has finished, the generated argumentation diagram is opened into the OpenCert Assurance Case Editor (see Figure 69). The target argumentation model and the related diagram are saved locally in a new project into the current Workspace under the name "Argumentation", and can be found in the "Package Explorer" tab, highlighted in red in Figure 69.

The generated argumentation model and diagram are also stored in the corresponding destination assurance case in the CDO Repository under the "ARGUMENTATION" folder (see Figure 70). To open the Repository Explorer , click "Window → Show View → Other" and under the OpenCert folder click on "Repository Explorer". To visualize the diagram stored in the CDO repository, double click on the Argumentation Diagram file (.arg_diagram) to open a diagram in the editing window, select the generated elements from the Outline View and drag them into the editing space, see Figure 70. The elements will be shown in the editor.


Project Explorer
Repository Ex...
CS11-Proc-Arg clean
ASSURANCE_PROJECT clean
ARGUMENTATION clean
argModel.arg clean
argDiagram.arg_diagram clean
Outline
Templates ...
Reuse
type filter text
Claim TaskDescriptor: develop_software_
Argument Reasoning Argument over role
Claim Role: aocs_sw_vv_manager transie
Asserted Inference Ard_Crd transient
Asserted Evidence Crd_Erd transient
Information Element Citation aocs_sw_vv
Asserted Evidence Crd_Erd transient
Information Element Citation aocs_sw_vv
Asserted Evidence Crd_Erd transient
Information Element Citation aocs_sw_vv
Asserted Evidence Crd_Erd transient
Information Element Citation aocs_sw_vv
Asserted Evidence Crd_Erd transient
Information Element Citation aocs_sw_vv
Asserted Evidence Crd_Erd transient
Information Element Citation aocs_sw_vv
Asserted Evidence Crd_Erd transient
Information Element Citation aocs_sw_vv
Claim Role: aocs_ait transient
Asserted Inference Ard_Crd transient
argDiagram.arg_diagram
G204.T_GZx...
The supplier s hall comple...
S92.R_GZx...
Argument over role...
G207.R_GZx...
development_ team_leader...
Drag and Drop
AssertedEvidence
AssertedEvidence
AssertedEvidence
AssertedEvidence
E570.R_G...
Develop...
E571.R_G...
Develop...
E572.R_G...
Develop...
E573.R_G...
Develop...
argModel.arg
Resource Set
Case
Goal ECSS-E-ST-40_Planning_Phase
Argument Reasoning Argument ove
Asserted Inference CP_Arph
Goal design of software items
Asserted Inference Arph_Phcl
Context design of software items's c
Asserted Context Ph_Con
Argument Reasoning Argument ove
Asserted Inference Ph_Aract
Goal Activity: Develop and documer
Asserted Inference Aph_Act
Argument Reasoning Argument ove
Asserted Inference Act_Artask
Asserted Inference Act_Td
Goal TaskDescriptor: development_
Argument Reasoning Argument ove
Goal Role: aocs_ait
Asserted Inference Ard_Crd
Asserted Evidence Crd_Erd
Solution aocs_ait's certification
Asserted Evidence Crd_Erd
Selection
Parent
List
Tree
Table
Properties
InformationElementCitation
Base
Appearance
Properties

Id: E573.R_GZxmgX1hEei51cvLnUNfhA
Description: Development Team Leader's certification against good analytical and problem-solving skills.

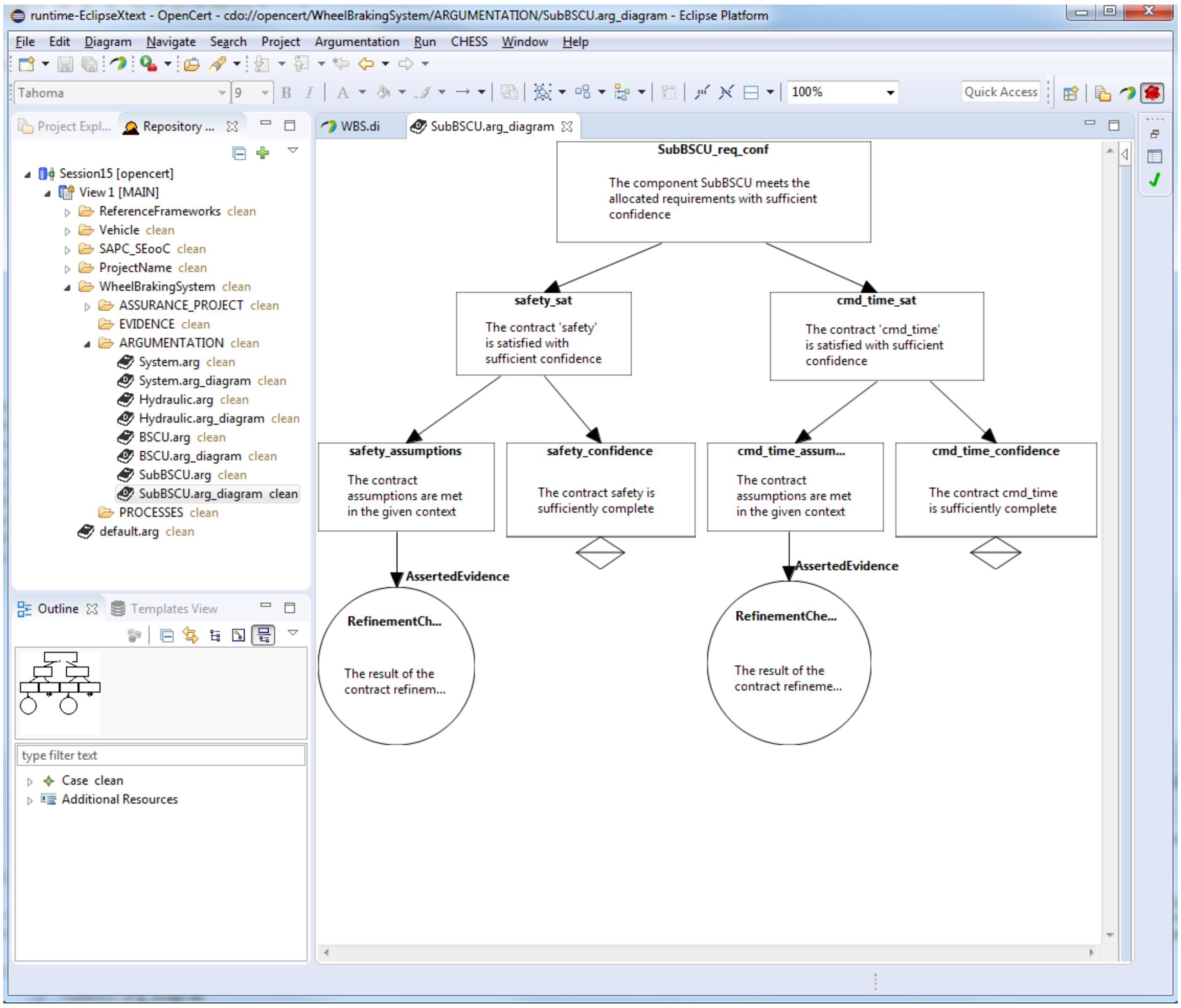
Figure 70. Generated argumentation model and diagram in CDO Repository

The proposed solution only focuses on the omission of key evidence fallacies in process-based argumentation. Although this improves the quality and the correctness of the generated arguments, there may exist other types of fallacies. Therefore, the generated arguments will be checked manually for other types of fallacies and eventually corrected (if other fallacies are detected). The fallacy detection process took a total time around a second to detect omission of key evidence fallacies in AOCS, while generation of process-based argumentation process took few minutes which is quite reasonable as compared to manually create argument.

3.4.2 Automatic generation of product-based arguments

The detailed guidelines for the automatic generation of product-based arguments are presented in the AMASS deliverable D3.8 [5]. The focus of this section is on the reuse aspect of the automatic product-based arguments generation. The *argumentGenerator* plugin has been implemented as part of OpenCert to act as a bridge between CHESS [26] modelling environment and assurance case modelling environment. Each component in CHESS can be described using assumption/guarantee contracts. The first step in ensuring that a component can be reused is ensuring that the contracts have been validated in the current system.

To facilitate reuse of the knowledge of the components, the CHESS metamodel has been extended to allow associating the knowledge of the component to the corresponding contract so that the knowledge of the component can be reused if the contract is validated. The *argumentGenerator* plugin prompts the user to select the CHESS model for which an argument should be generated, as well as the target assurance case model. The plugin generates a set of product-based argument fragments for each of the components in the source model by including knowledge of the component associated to the validated contracts in the argument-fragment. An example of a generated argument-fragment is shown in Figure 71.



SubBSCU_req_conf: The component SubBSCU meets the allocated requirements with sufficient confidence

safety_sat: The contract 'safety' is satisfied with sufficient confidence

cmd_time_sat: The contract 'cmd_time' is satisfied with sufficient confidence

safety_assumptions: The contract assumptions are met in the given context

safety_confidence: The contract safety is sufficiently complete

cmd_time_assum...: The contract assumptions are met in the given context

cmd_time_confidence: The contract cmd_time is sufficiently complete

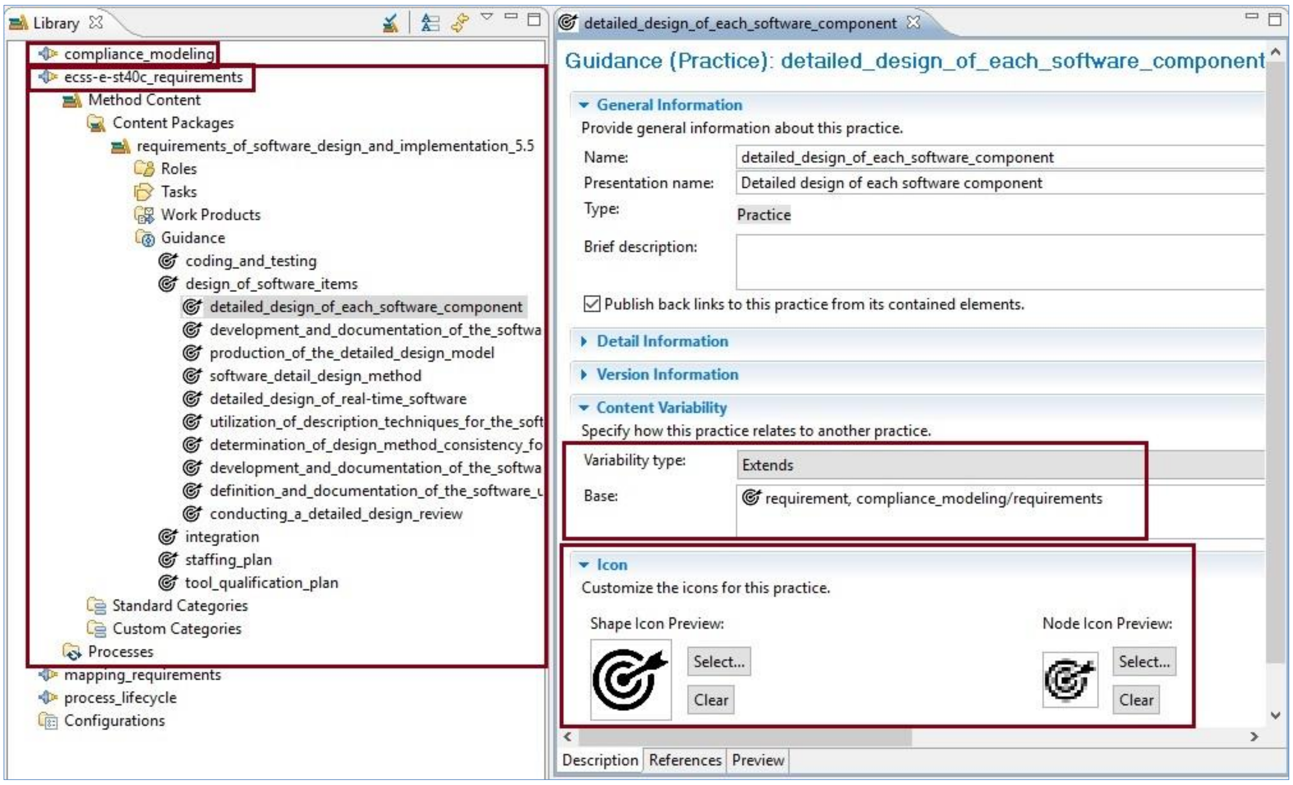
RefinementCh...: The result of the contract refinem...

RefinementChe...: The result of the contract refineme...

Figure 71. Example of a generated product-based argument-fragment

3.5 Transformation of Standard's Requirement from EPF-C to Baseline Model OpenCert (*)

Standards provide the guidance, in the form of processes to be followed during the development of safety-critical systems. EPF-C is used for the modelling of standard's requirements, planning of processes and basic compliance via a mapping as described in user manual [2]. For modelling standard requirements in the *requirements* plugin, the guidance type "Practice" has been customized with an icon and variability relationship "Extends" with *compliance_modelling* plugin, as shown in Figure 72. In order to define the mapping, standard requirements are copied in the *mapping_requirements* plugin. These copied requirements have a variability relationship “Contributes” with original requirements, as shown in Figure 73. In addition, the links between requirements and lifecycle elements have been established through “References”, as shown in Figure 74.


Library
compliance_modeling
ecss-e-st40c_requirements
Method Content
Content Packages
requirements_of_software_design_and_implementation_5.5
Roles
Tasks
Work Products
Guidance
coding_and_testing
design_of_software_items
detailed_design_of_each_software_component
development_and_documentation_of_the_softwa
production_of_the_detailed_design_model
software_detail_design_method
detailed_design_of_real-time_software
utilization_of_description_techniques_for_the_soft
determination_of_design_method_consistency_fo
development_and_documentation_of_the_softwa
definition_and_documentation_of_the_software_u
conducting_a_detailed_design_review
integration
staffing_plan
tool_qualification_plan
Standard Categories
Custom Categories
Processes
mapping_requirements
process_lifecycle
Configurations

detailed_design_of_each_software_component

Guidance (Practice): detailed_design_of_each_software_component

General Information
Provide general information about this practice.
Name: detailed_design_of_each_software_component
Presentation name: Detailed design of each software component
Type: Practice
Brief description:
☑ Publish back links to this practice from its contained elements.

Detail Information

Version Information

Content Variability
Specify how this practice relates to another practice.
Variability type: Extends
Base: requirement, compliance_modeling/requirements

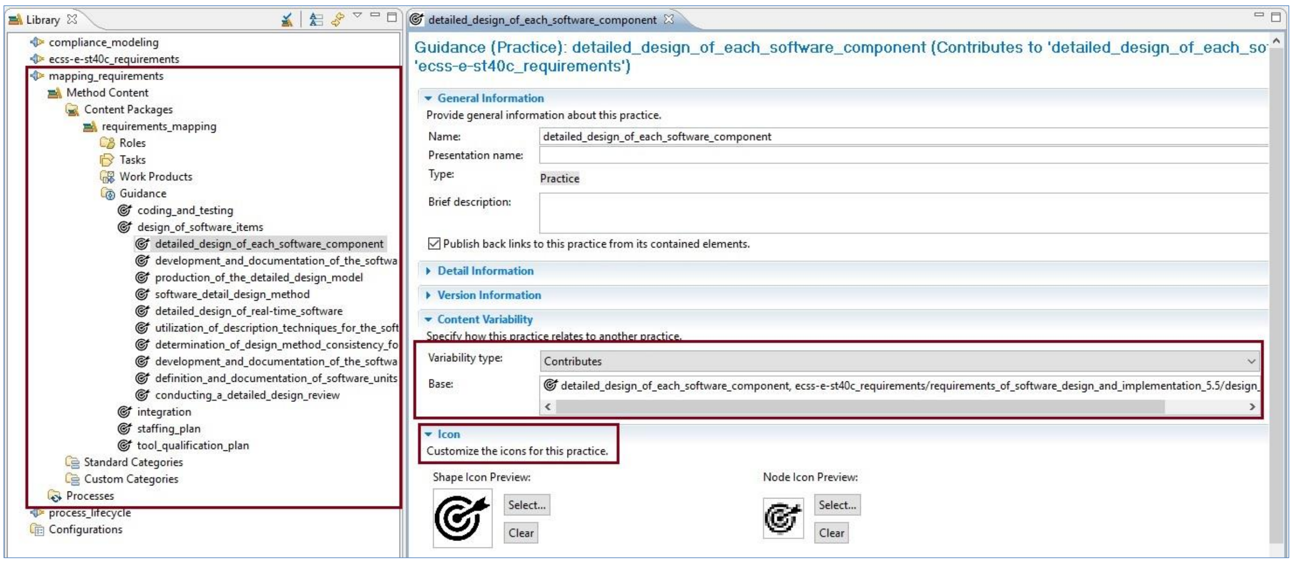
Icon
Customize the icons for this practice.
Shape Icon Preview: Select... Clear
Node Icon Preview: Select... Clear

Description | References | Preview

Figure 72. Modelling requirements

The OpenCert tool allows the users to model baselines as requirements. Instead of creating the Baseline model from stretch, a first view of model can be achieved by transforming the standard requirements modelled in EPF-C into OpenCert. The requirements transformation takes the requirements modelled in EPF-C as “Practice” under the Content Package (see Figure 75) and generates the Baseline model and diagram. The main mapping rules for baseline generation are as follows:

- Content Package of EPF-C related to requirements is mapped into Base Framework.
- Requirements (modelled as practices) are mapped into Base Activity.
- Requirements (modelled as sub-practices) are mapped into Base Requirements.
- Id, name and description of requirements are mapped into Id, name and description.


Library

- compliance_modeling
- ecss-e-st40c_requirements
- mapping_requirements
  - Method Content
    - Content Packages
      - requirements_mapping
        - Roles
        - Tasks
        - Work Products
        - Guidance
          - coding_and_testing
          - design_of_software_items
            - detailed_design_of_each_software_component
            - development_and_documentation_of_the_softwa
            - production_of_the_detailed_design_model
            - software_detail_design_method
            - detailed_design_of_real-time_software
            - utilization_of_description_techniques_for_the_soft
            - determination_of_design_method_consistency_fo
            - development_and_documentation_of_the_softwa
            - definition_and_documentation_of_software_units
            - conducting_a_detailed_design_review
          - integration
          - staffing_plan
          - tool_qualification_plan
    - Standard Categories
    - Custom Categories
  - Processes
- process_lifecycle
- Configurations

detailed_design_of_each_software_component

Guidance (Practice): detailed_design_of_each_software_component (Contributes to 'detailed_design_of_each_so 'ecss-e-st40c_requirements')

General Information
Provide general information about this practice.
Name: detailed_design_of_each_software_component
Presentation name:
Type: Practice
Brief description:
☑ Publish back links to this practice from its contained elements.

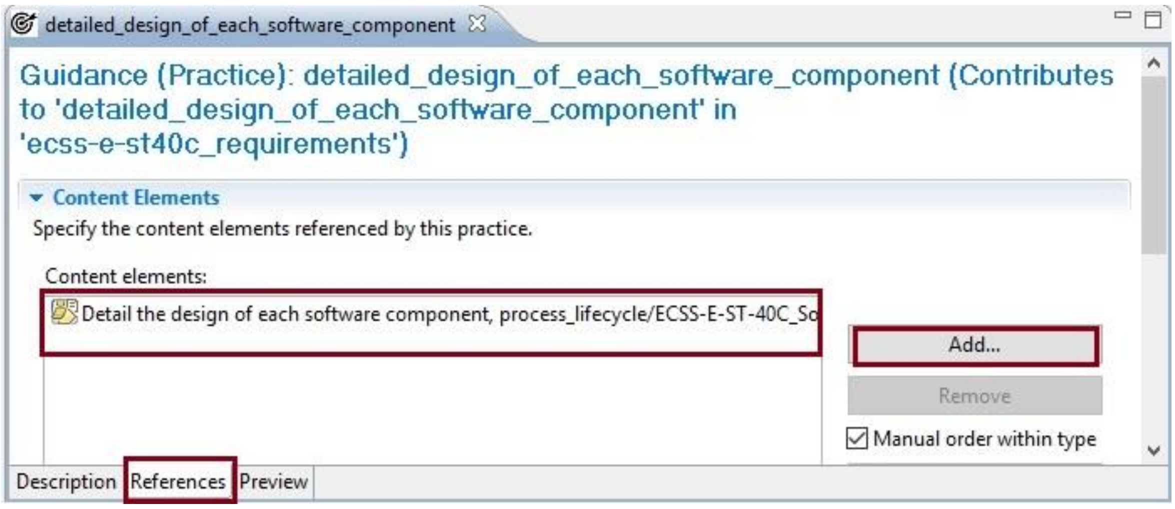
Detail Information

Version Information

Content Variability
Specify how this practice relates to another practice.
Variability type: Contributes
Base: detailed_design_of_each_software_component, ecss-e-st40c_requirements/requirements_of_software_design_and_implementation_5.5/design_

Icon
Customize the icons for this practice.
Shape Icon Preview: Select... Clear
Node Icon Preview: Select... Clear

Figure 73. Mapping Requirements


detailed_design_of_each_software_component

Guidance (Practice): detailed_design_of_each_software_component (Contributes to 'detailed_design_of_each_software_component' in 'ecss-e-st40c_requirements')

Content Elements
Specify the content elements referenced by this practice.
Content elements:
Detail the design of each software component, process_lifecycle/ECSS-E-ST-40C_So

Add...
Remove
☑ Manual order within type

Description | References | Preview

Figure 74. Add references to process elements

To generate the Baseline model, the following steps need to be performed:

1. Invoke the Baseline Generator plugin by clicking on the Content Package, in particular, by selecting the "Transformation -> Generate Baseline Model" option, as shown in Figure 75.

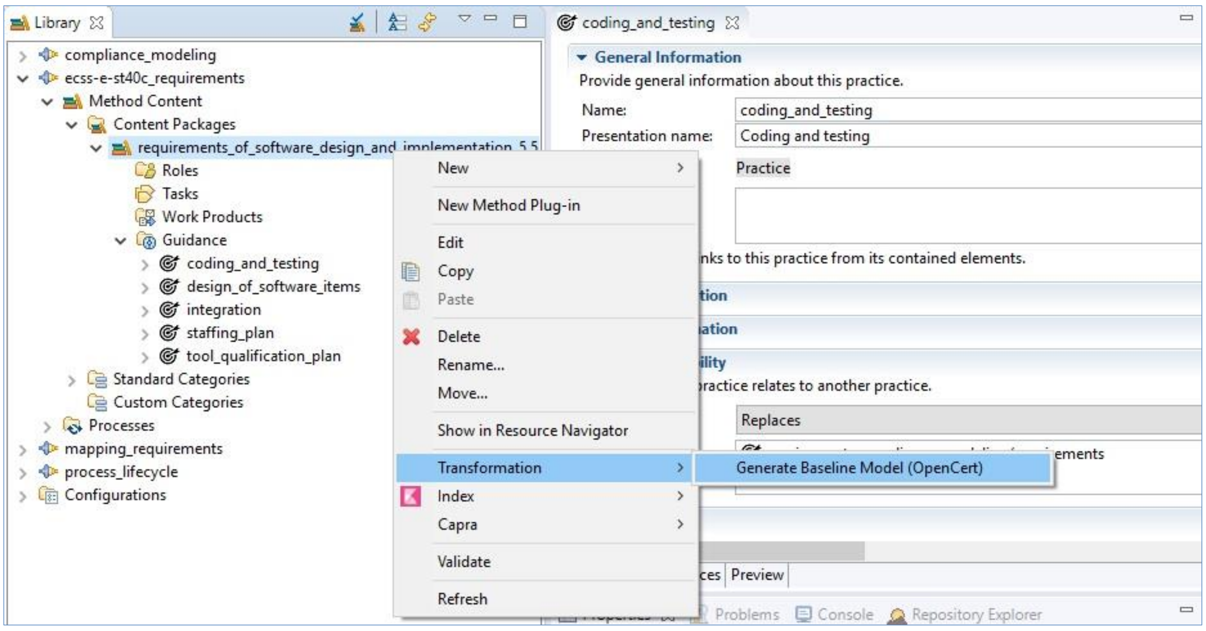
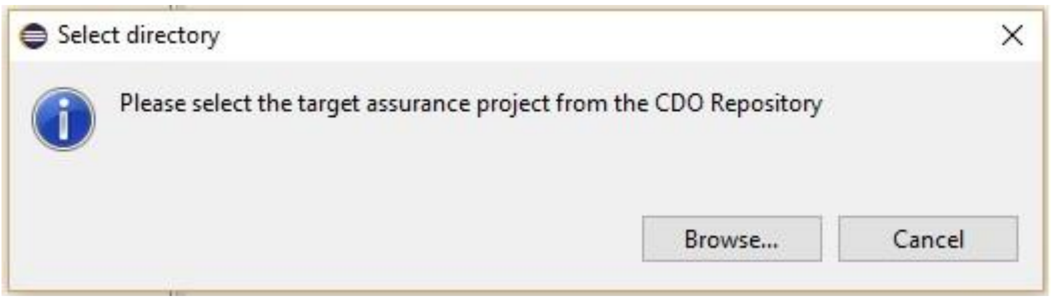


Figure 75. Transform Requirements into Baseline Model

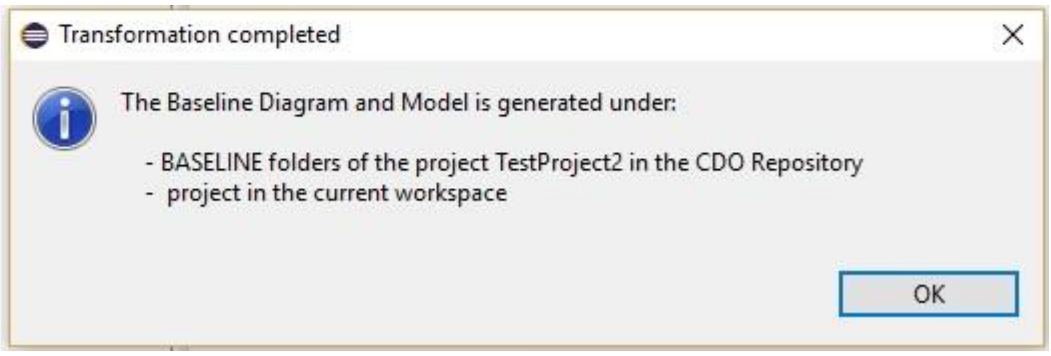
2. Select the target Assurance Project from the CDO Repository using the “Browse” functionality (see Figure 76). Please note that the CDO Repository should have already been configured and Repository Explorer must be opened. For this, click on “Window → Preferences”. Then, in the Preferences window, open the “OpenCert” menu, click on “Model Repository Preferences”, and fill the required fields.


Select directory

Please select the target assurance project from the CDO Repository

Figure 76. Browsing the Target Assurance Project in CDO repository

3. The dialogue in Figure 77 appears to inform that the transformation process has successfully finished. Press the “OK” button to finish the generation process.

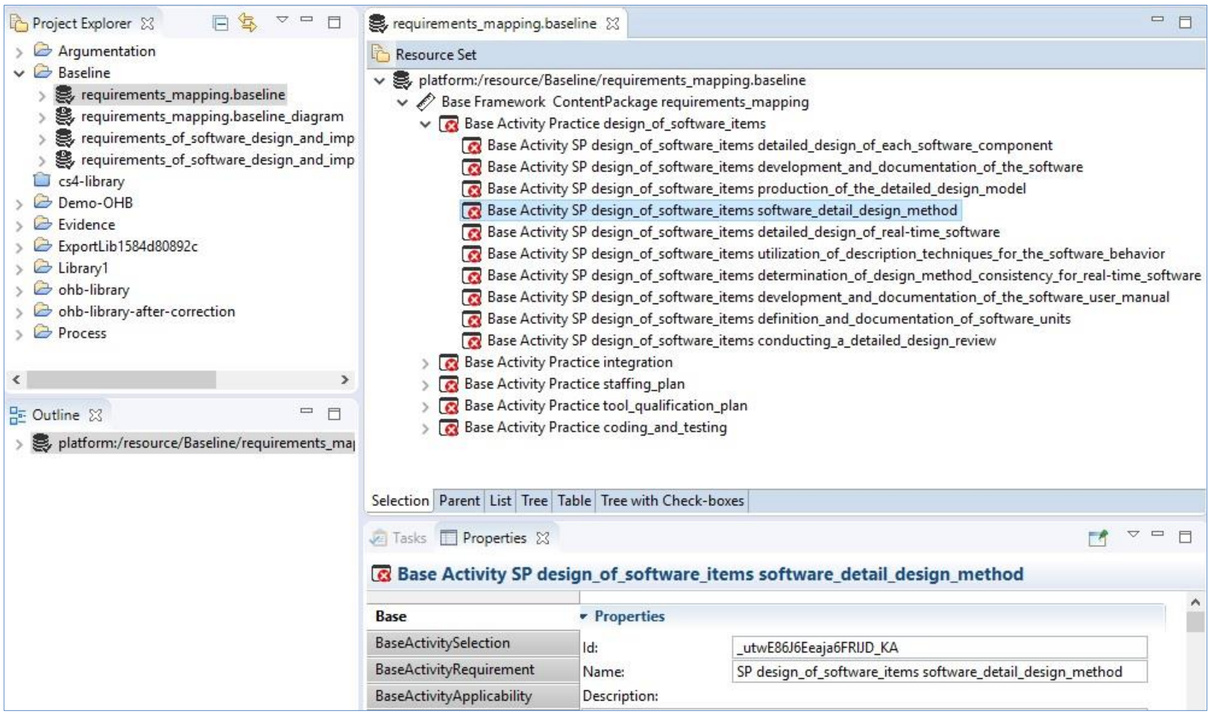

Transformation completed

The Baseline Diagram and Model is generated under:

- BASELINE folders of the project TestProject2 in the CDO Repository
- project in the current workspace

Figure 77. Transformation Completed

4. Figure 78 and Figure 79 shows the generated Baseline model and diagram respectively.



Project Explorer

- Argumentation
- Baseline
  - requirements_mapping.baseline
  - requirements_mapping.baseline_diagram
  - requirements_of_software_design_and_imp
  - requirements_of_software_design_and_imp
  - cs4-library
- Demo-OHB
- Evidence
- ExportLib1584d80892c
- Library1
- ohb-library
- ohb-library-after-correction
- Process

Outline

- platform:/resource/Baseline/requirements_ma

requirements_mapping.baseline

Resource Set

- platform:/resource/Baseline/requirements_mapping.baseline
  - Base Framework ContentPackage requirements_mapping
    - Base Activity Practice design_of_software_items
      - Base Activity SP design_of_software_items detailed_design_of_each_software_component
      - Base Activity SP design_of_software_items development_and_documentation_of_the_software
      - Base Activity SP design_of_software_items production_of_the_detailed_design_model
      - Base Activity SP design_of_software_items software_detail_design_method
      - Base Activity SP design_of_software_items detailed_design_of_real-time_software
      - Base Activity SP design_of_software_items utilization_of_description_techniques_for_the_software_behavior
      - Base Activity SP design_of_software_items determination_of_design_method_consistency_for_real-time_software
      - Base Activity SP design_of_software_items development_and_documentation_of_the_software_user_manual
      - Base Activity SP design_of_software_items definition_and_documentation_of_software_units
      - Base Activity SP design_of_software_items conducting_a_detailed_design_review
    - Base Activity Practice integration
    - Base Activity Practice staffing_plan
    - Base Activity Practice tool_qualification_plan
    - Base Activity Practice coding_and_testing

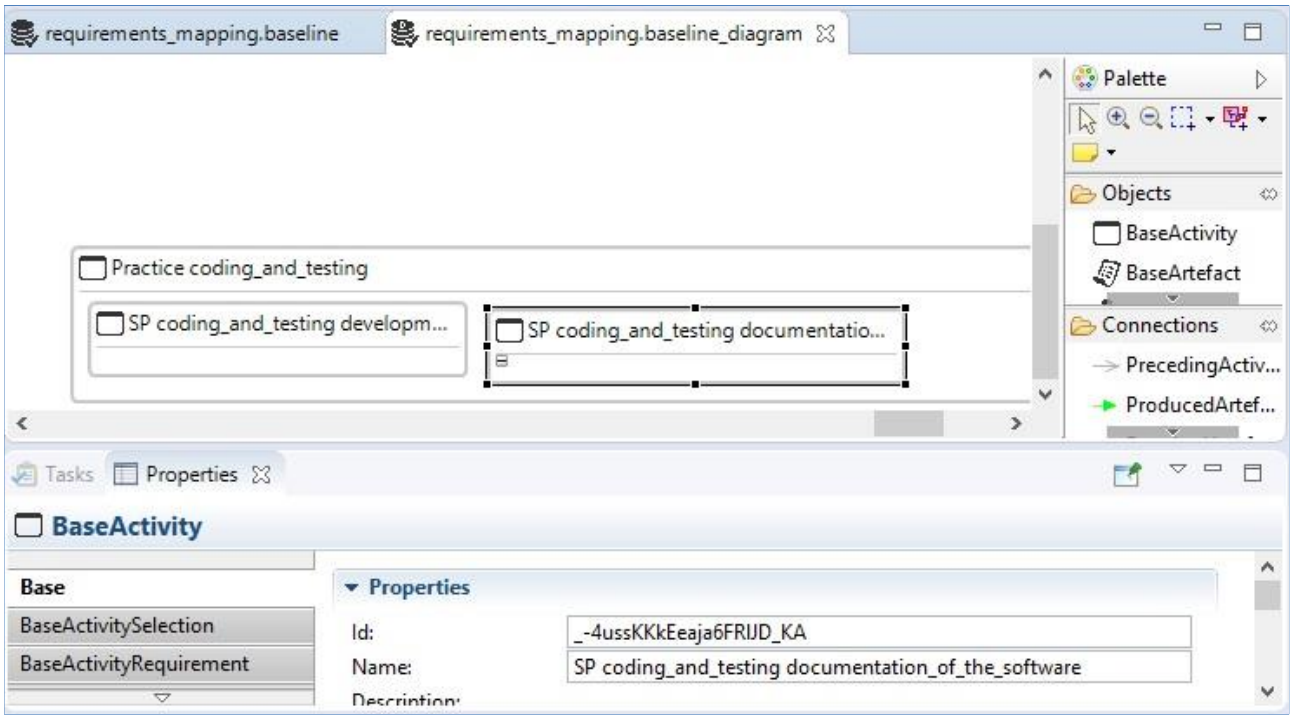
Selection | Parent | List | Tree | Table | Tree with Check-boxes

Tasks | Properties

Base Activity SP design_of_software_items software_detail_design_method

Base	Properties	
BaseActivitySelection	Id:	_utwE86J6Eeaja6FRIJD_KA
BaseActivityRequirement	Name:	SP design_of_software_items software_detail_design_method
BaseActivityApplicability	Description:	

Figure 78. Generated Baseline Model



requirements_mapping.baseline | requirements_mapping.baseline_diagram

Practice coding_and_testing

SP coding_and_testing developm...

SP coding_and_testing documentatio...

Palette

Objects
- BaseActivity
- BaseArtefact

Connections
- PrecedingActiv...
- ProducedArtef...

Tasks | Properties

BaseActivity

Base	Properties	
BaseActivitySelection	Id:	_-4ussKKkEeaja6FRIJD_KA
BaseActivityRequirement	Name:	SP coding_and_testing documentation_of_the_software
	Description:	

Figure 79. Generated Baseline Diagram

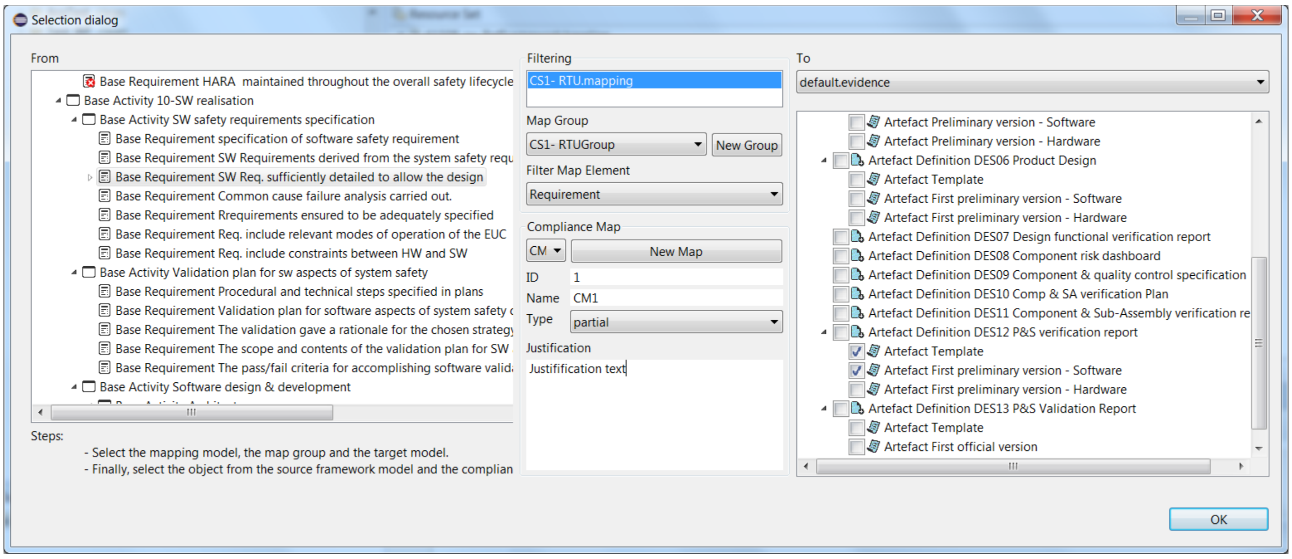
3.6 Compliance Mapping (*)

The AMASS Platform help “engineers” to assess where they are with respect to their duties to conform to safety practices and standards, and still to motivate them to see the effective progress of the work and level of compliance.

The Compliance Mapping functionality objective is to create maps between the information from the standards (i.e. obligations) with the information managed in the context of a given assurance project (i.e. accomplishment).

There are two features related with the compliance mapping management. One related to monitoring the compliance status of an assurance project (see Figure 81) and another one to create or modify an assurance project's compliance maps (see Figure 80).

The user is referred to the user manual [2] to get a deeper knowledge about these specific options provided by the AMASS tools, concretely the section 6.4.3 about monitoring the compliance status and the section 6.4.2. about editing the compliance maps.


Selection dialog

From

- Base Requirement HARA maintained throughout the overall safety lifecycle
- Base Activity 10-SW realisation
  - Base Activity SW safety requirements specification
    - Base Requirement specification of software safety requirement
    - Base Requirement SW Requirements derived from the system safety requ
    - Base Requirement SW Req. sufficiently detailed to allow the design
    - Base Requirement Common cause failure analysis carried out.
    - Base Requirement Rrequirements ensured to be adequately specified
    - Base Requirement Req. include relevant modes of operation of the EUC
    - Base Requirement Req. include constraints between HW and SW
  - Base Activity Validation plan for sw aspects of system safety
    - Base Requirement Procedural and technical steps specified in plans
    - Base Requirement Validation plan for software aspects of system safety
    - Base Requirement The validation gave a rationale for the chosen strategy
    - Base Requirement The scope and contents of the validation plan for SW
    - Base Requirement The pass/fail criteria for accomplishing software valid
  - Base Activity Software design & development

Steps:

- Select the mapping model, the map group and the target model.
- Finally, select the object from the source framework model and the complian

Filtering: CS1- RTU.mapping

Map Group: CS1- RTUGroup [New Group]

Filter Map Element: Requirement

Compliance Map: CM [New Map]

ID: 1

Name: CM1

Type: partial

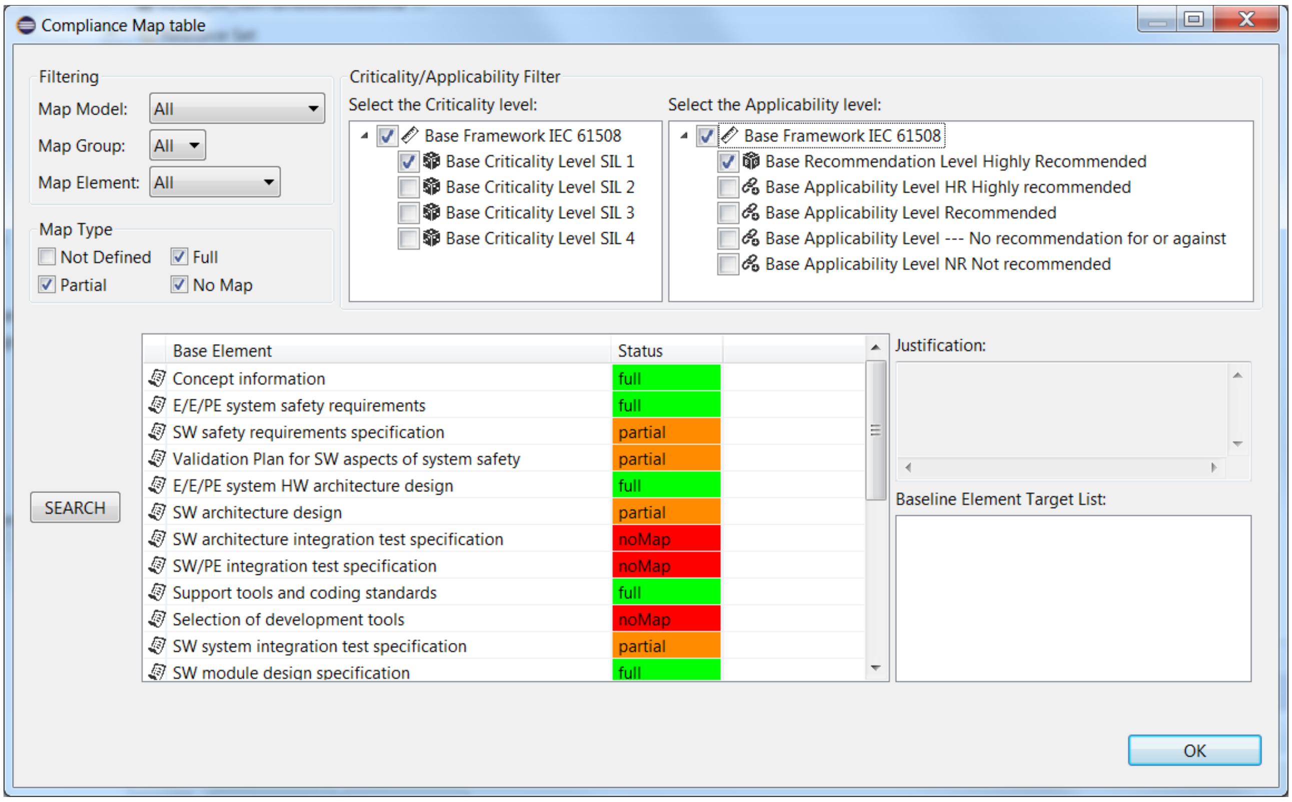
Justification: Justification text

To: default.evidence

- ☐ Artefact Preliminary version - Software
- ☐ Artefact Preliminary version - Hardware
- ☐ Artefact Definition DES06 Product Design
  - ☐ Artefact Template
  - ☐ Artefact First preliminary version - Software
  - ☐ Artefact First preliminary version - Hardware
- ☐ Artefact Definition DES07 Design functional verification report
- ☐ Artefact Definition DES08 Component risk dashboard
- ☐ Artefact Definition DES09 Component & quality control specification
- ☐ Artefact Definition DES10 Comp & SA verification Plan
- ☐ Artefact Definition DES11 Component & Sub-Assembly verification re
- ☐ Artefact Definition DES12 P&S verification report
  - ☑ Artefact Template
  - ☑ Artefact First preliminary version - Software
  - ☐ Artefact First preliminary version - Hardware
- ☐ Artefact Definition DES13 P&S Validation Report
  - ☐ Artefact Template
  - ☐ Artefact First official version

[OK]

Figure 80. Compliance mapping editor


Compliance Map table

Filtering

Map Model: All

Map Group: All

Map Element: All

Map Type: ☐ Not Defined ☑ Full ☑ Partial ☑ No Map

Criticality/Applicability Filter

Select the Criticality level:

- ☑ Base Framework IEC 61508
  - ☑ Base Criticality Level SIL 1
  - ☐ Base Criticality Level SIL 2
  - ☐ Base Criticality Level SIL 3
  - ☐ Base Criticality Level SIL 4

Select the Applicability level:

- ☑ Base Framework IEC 61508
  - ☑ Base Recommendation Level Highly Recommended
  - ☐ Base Applicability Level HR Highly recommended
  - ☐ Base Applicability Level Recommended
  - ☐ Base Applicability Level --- No recommendation for or against
  - ☐ Base Applicability Level NR Not recommended

[SEARCH]

Base Element	Status
Concept information	full
E/E/PE system safety requirements	full
SW safety requirements specification	partial
Validation Plan for SW aspects of system safety	partial
E/E/PE system HW architecture design	full
SW architecture design	partial
SW architecture integration test specification	noMap
SW/PE integration test specification	noMap
Support tools and coding standards	full
Selection of development tools	noMap
SW system integration test specification	partial
SW module design specification	full

Justification:

Baseline Element Target List:

[OK]

Figure 81. Compliance mapping monitor

3.7 Compliance Checking (*)

This section provides the methodological guide for process compliance checking within AMASS. A process engineer deals with the representation of complex processes to be adopted to engineer the organization's systems. In addition, the process engineer must manage the compliance of these processes against domain-specific safety standards. Process compliance management is time-consuming and prone-to error, since the attention of the process engineer has to focus on checking that all the process-related requirements imposed by the standards are fulfilled within the process's entities. Within AMASS, a solution for automatic compliance checking is presented. This solution is based on the combination of EPF-C [25], which provides process modelling capabilities and Regorous [51], which provides compliance checking capabilities. The automated compliance checking vision is presented in Figure 82.

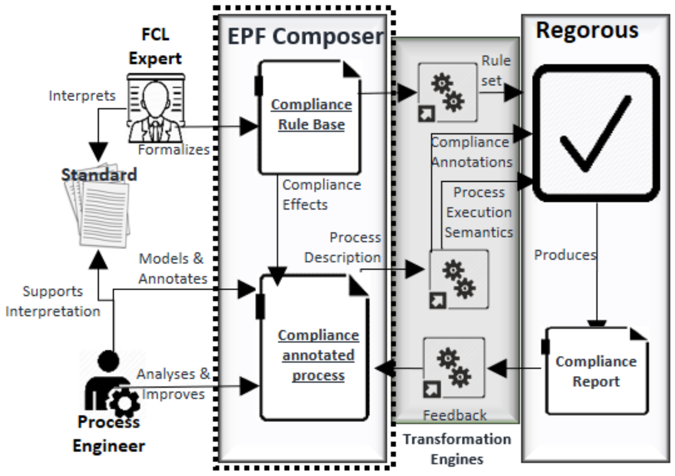


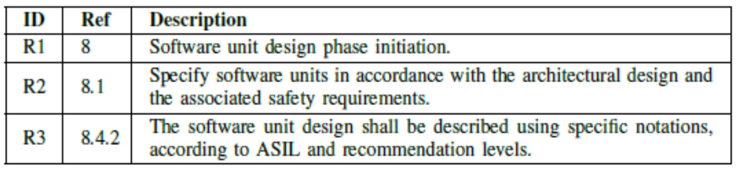
Figure 82. Automated Compliance Checking Vision [52]

The compliance checking vision includes two roles. The first one is the Process engineer, which should support the interpretation of the standard's requirements, model, annotate the process, and analyze the compliance report. The second is the FCL expert, which should interpret the standard's requirements and formalize them into Formal Contract Logic (FCL) [53].

3.7.1 Mechanisms to annotate software process models

This section focuses on the mechanisms required to annotate software process models in EPF-C (the area surrounded by the dotted line in Figure 82). For this, three plugins are created.

1. **Plugin for capturing standard's requirements:** In the method authoring of EPF-C, the standard's requirements are captured using custom categories. The root of the custom categories is the name of the standard. Then, the rule set is added using a customized reusable asset, the compliance effects using customized concepts, and the compliance effects are associated to the standard's requirements. As an example, the modelling of the standards requirements extracted from ISO 26262 part 6 (described in Figure 83) is presented below.



ID	Ref	Description
R1	8	Software unit design phase initiation.
R2	8.1	Specify software units in accordance with the architectural design and the associated safety requirements.
R3	8.4.2	The software unit design shall be described using specific notations, according to ASIL and recommendation levels.

Figure 83. Requirements for ISO 26262 part 6 clause 8

The standards requirements required to be formalized before they are modelled in EPF composer. Figure 84 presents the requirements in the formal format required by Regorous, namely FCL. The formalization process used in this example is in detail explained in [54].

$$\begin{aligned}
r_1 &:\Rightarrow [OM] addressSwUnitDesignProcess \\
r_2 &: addressSwUnitDesignProcess \\
&\Rightarrow [OANPNP] - performSpecifySwUnit \\
r'_2 &: performProvideSwArchitecturalDesign, \\
&performProvideSwSafetyRequirements \\
&\Rightarrow [P] performSpecifySwUnit \\
r_3 &: performSpecifySwUnit \\
&\Rightarrow [OANPNP] selectMandatoryNotationsforSwDesign \\
r'_3 &: provideRationaleForNotSelectMandatoryNotationsforSwDesign \\
&\Rightarrow [P] - selectMandatoryNotationsforSwDesign \\
r'_2 &> r_2 \\
r'_3 &> r_3
\end{aligned}$$

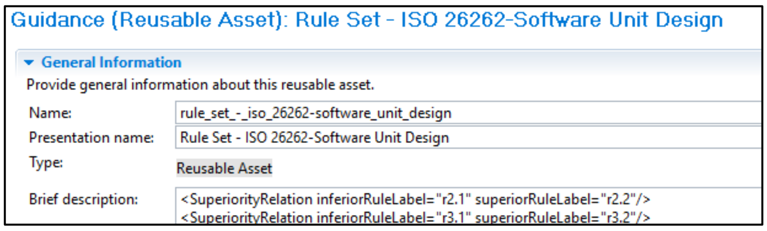
Figure 84. FCL rule set for ISO 26262 requirements [54]

In this plugin, the user first defines the compliance effects as well as the rule set in the guidance part of the Method content of the plugin, as presented in Figure 87, box red. For modeling compliance effects, which can be extracted from the formulas of the rules (precedent and consequent of the rules), customized concepts are used. For the rule set, a reusable asset is used. The customization mechanism is presented in Figure 85.

EPF-C	Compliance Information	Suggested Icons
Reusable Asset	**Rule Set**	▤
-URL/attachment	-uri	-
-presentation name	-display Name	-
Concept	**Compliance Effect**	➡
-name	-effect name	-
-brief description	-effect description	-
Custom category	**Standard requirement**	🎯
-name	-short name	-
-brief description	-requirement description	-

Figure 85. EPF-C customization

The rule set (create with a reusable asset) contains the information related to the superiority relations (see Figure 86).


Guidance (Reusable Asset): Rule Set - ISO 26262-Software Unit Design

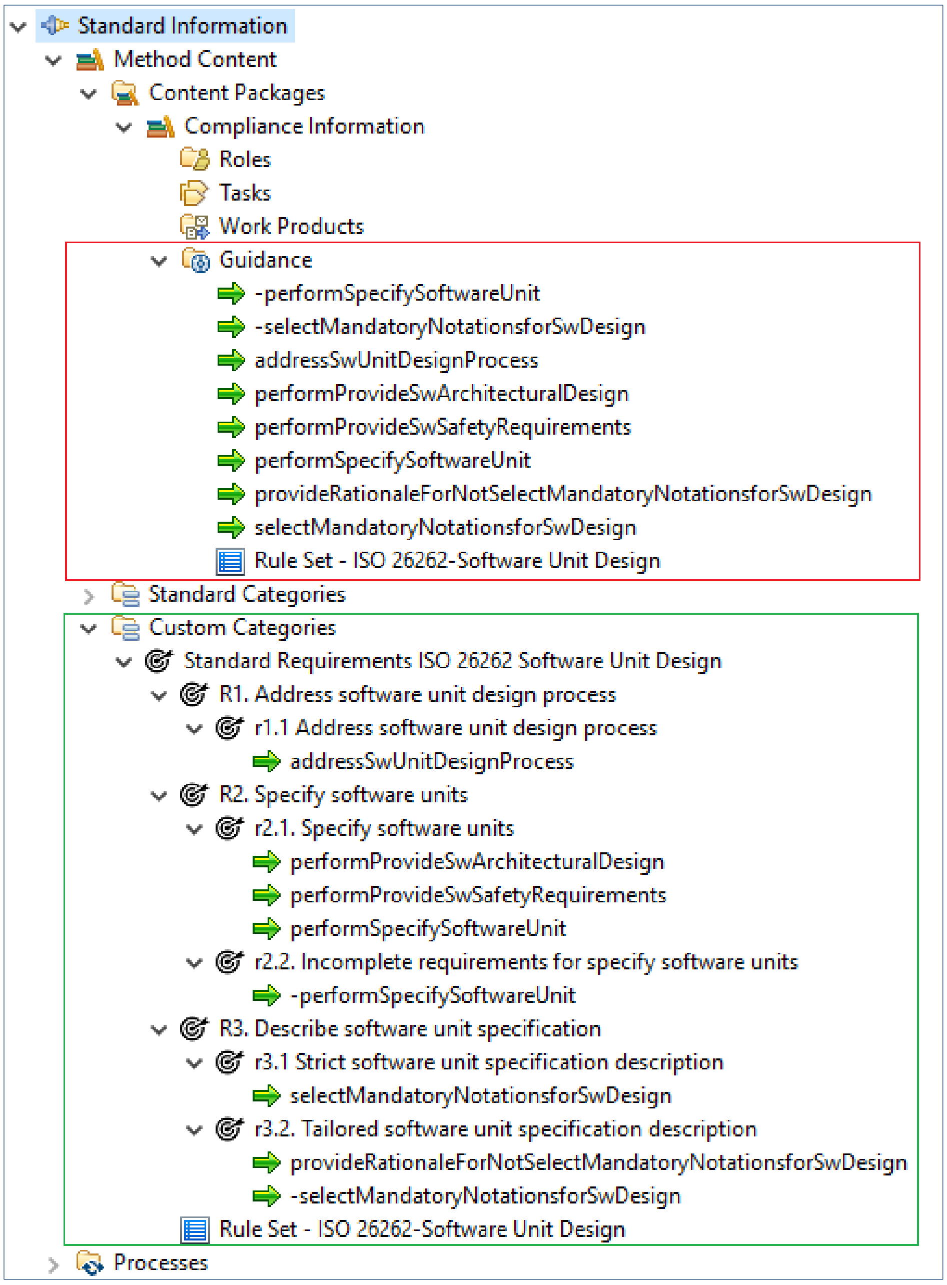
General Information

Provide general information about this reusable asset.

Name:	rule_set_-_iso_26262-software_unit_design
Presentation name:	Rule Set - ISO 26262-Software Unit Design
Type:	Reusable Asset
Brief description:	<SuperiorityRelation inferiorRuleLabel="r2.1" superiorRuleLabel="r2.2"/> <SuperiorityRelation inferiorRuleLabel="r3.1" superiorRuleLabel="r3.2"/>

Figure 86. Rule set definition

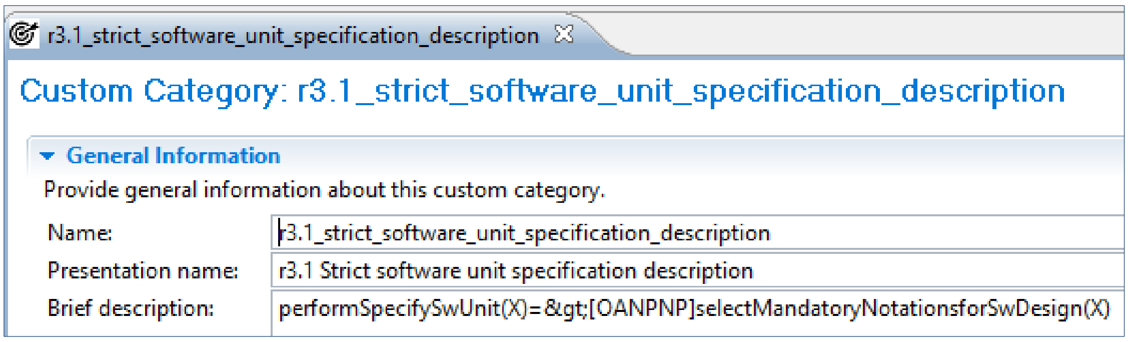
Once the compliance effects and the rule set are defined, they have to be connected with the rules and the requirements. Thus, in the custom category area (see the information surrounded by a green box in Figure 87) a category root is defined. It is named *standard_requirements_iso_26262_software_unit* and associated to the requirements presented in Figure 83, with a short but descriptive name, in a nested list of custom categories. Then, rules that are associated to the requirement are created. For example, R3 is a requirement that can be tailored. Therefore, R3 has two associated rules r3.1 and r3.2. Then, rules are associated to the corresponding compliance effects. The customized list of standard's requirements, the rules and compliance annotations are depicted in Figure 87.



Standard Information
- Method Content
  - Content Packages
    - Compliance Information
      - Roles
      - Tasks
      - Work Products
      - Guidance
        - -performSpecifySoftwareUnit
        - -selectMandatoryNotationsforSwDesign
        - addressSwUnitDesignProcess
        - performProvideSwArchitecturalDesign
        - performProvideSwSafetyRequirements
        - performSpecifySoftwareUnit
        - provideRationaleForNotSelectMandatoryNotationsforSwDesign
        - selectMandatoryNotationsforSwDesign
        - Rule Set - ISO 26262-Software Unit Design
  - Standard Categories
  - Custom Categories
    - Standard Requirements ISO 26262 Software Unit Design
      - R1. Address software unit design process
        - r1.1 Address software unit design process
          - addressSwUnitDesignProcess
      - R2. Specify software units
        - r2.1. Specify software units
          - performProvideSwArchitecturalDesign
          - performProvideSwSafetyRequirements
          - performSpecifySoftwareUnit
        - r2.2. Incomplete requirements for specify software units
          - -performSpecifySoftwareUnit
      - R3. Describe software unit specification
        - r3.1 Strict software unit specification description
          - selectMandatoryNotationsforSwDesign
        - r3.2. Tailored software unit specification description
          - provideRationaleForNotSelectMandatoryNotationsforSwDesign
          - -selectMandatoryNotationsforSwDesign
    - Rule Set - ISO 26262-Software Unit Design
- Processes

Figure 87. Standard's requirements plugin

The rule is written in the main description field of the compliance effect, e.g., see r3.1 in Figure 88 (the symbol => is represented by "**&gt**" in the rule definition).



Custom Category: r3.1_strict_software_unit_specification_description

General Information

Provide general information about this custom category.

Name:	r3.1_strict_software_unit_specification_description
Presentation name:	r3.1 Strict software unit specification description
Brief description:	performSpecifySwUnit(X)=&gt;[OANPNP]selectMandatoryNotationsforSwDesign(X)

Figure 88. Specification of a rule

2. **Plugin for capturing process elements**: In this example, the definition of the process elements is based on our interpretation of the requirements provided in Figure 83. From R1, it is deduced that there is one task called *Start Software Unit Design Process* in which the requirements for the process are collected, namely, the Software Architectural Design and the Software Safety requirements, which are work products resulting from previous phases. From R2 it is deduced the existence of the task *Specify Software Unit* and from R3, it can be deduced that there is a task called *Design Software Unit* which output is the Software Unit Design. The information related to this plugin is presented in Figure 89.

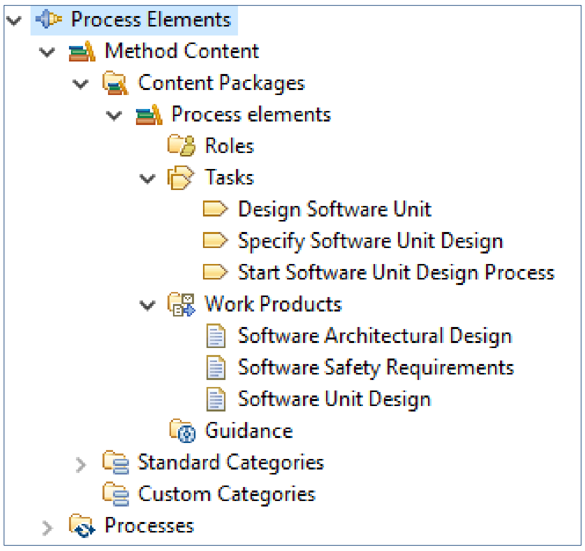

- Process Elements
  - Method Content
    - Content Packages
      - Process elements
        - Roles
        - Tasks
          - Design Software Unit
          - Specify Software Unit Design
          - Start Software Unit Design Process
        - Work Products
          - Software Architectural Design
          - Software Safety Requirements
          - Software Unit Design
        - Guidance
    - Standard Categories
    - Custom Categories
  - Processes

Figure 89. Process elements plugin

3. **Plugin for annotating process description**: a plugin is created, in which the tasks defined in the process elements plugin are copied (see the tasks surrounded by the red box in Figure 90). These tasks are extended with the variability type contributes in the description of the task (see Figure 91).

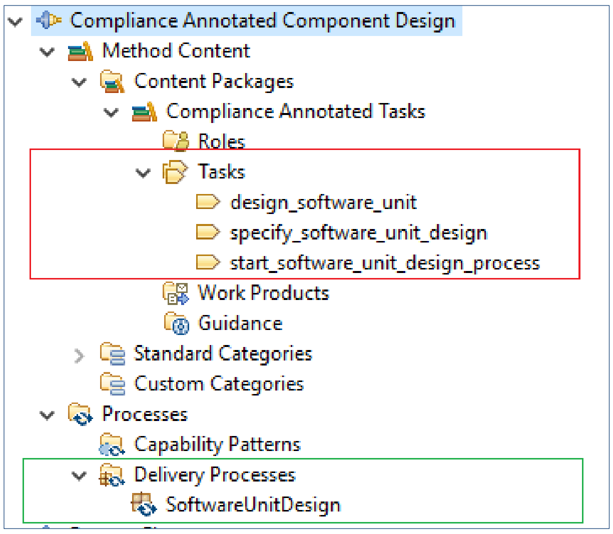
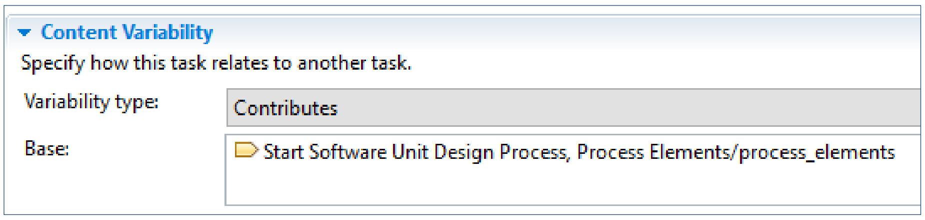

- Compliance Annotated Component Design
  - Method Content
    - Content Packages
      - Compliance Annotated Tasks
        - Roles
        - Tasks
          - design_software_unit
          - specify_software_unit_design
          - start_software_unit_design_process
        - Work Products
        - Guidance
    - Standard Categories
    - Custom Categories
  - Processes
    - Capability Patterns
    - Delivery Processes
      - SoftwareUnitDesign

Figure 90. Plugin for the annotated process

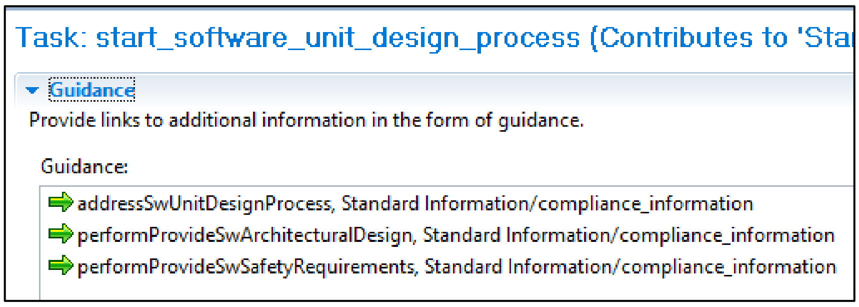

Content Variability

Specify how this task relates to another task.

Variability type:	Contributes
Base:	Start Software Unit Design Process, Process Elements/process_elements

Figure 91. Mechanism for extending a task

Then, the tasks are annotated by deducing the compliance effects that they produce. For example, the task *Start Software Unit Design Process*, produces the compliance effect *addressSoftwareUnitDesignProcess*, since with this task the process is addressed. Also, this task has two inputs, i.e., the software safety requirements and the architectural design. Thus, this task is also producing the compliance effects *performProvideAssociatedSwSafetyRequirements* and *performProvideSwArchitecturalDesign*. The annotation can be seen in the section called Concepts (see Figure 92).


Task: start_software_unit_design_process (Contributes to 'Sta

Guidance

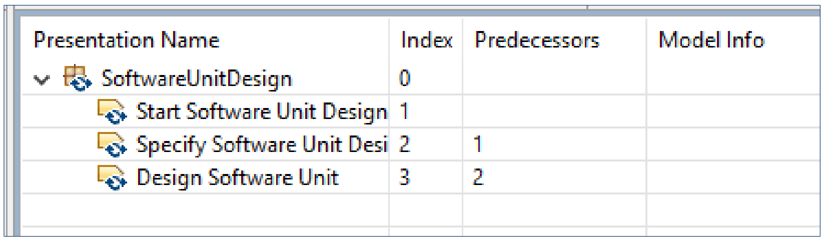
Provide links to additional information in the form of guidance.

Guidance:

- addressSwUnitDesignProcess, Standard Information/compliance_information
- performProvideSwArchitecturalDesign, Standard Information/compliance_information
- performProvideSwSafetyRequirements, Standard Information/compliance_information

Figure 92. Task annotation

Then, the delivery process is created (see Figure 90, the area surrounded by the green box). In the breakdown structure of the delivery process, the tasks that are required by the process are created (see Figure 93).



Presentation Name	Index	Predecessors	Model Info
SoftwareUnitDesign	0		
Start Software Unit Design	1		
Specify Software Unit Desi	2	1	
Design Software Unit	3	2	

Figure 93. Breakdown structure of the delivery process

Once the breakdown structure is defined, the activity diagram is created by selecting the option “Open Activity Diagram” from the contextual menu that appears after right-clicking on the root of the process (see Figure 94).

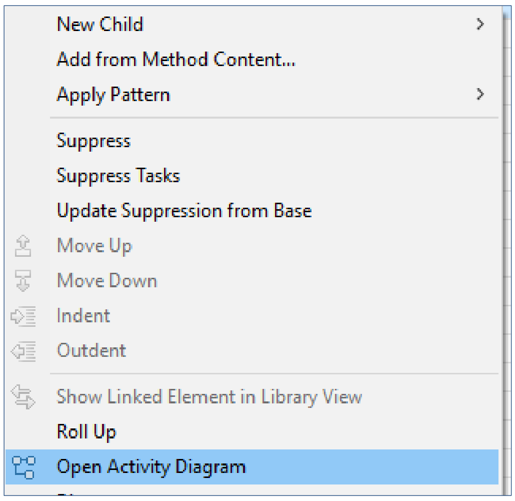


Figure 94. Option for create the activity diagram

The activity diagram would look similar to the one created in Figure 95.

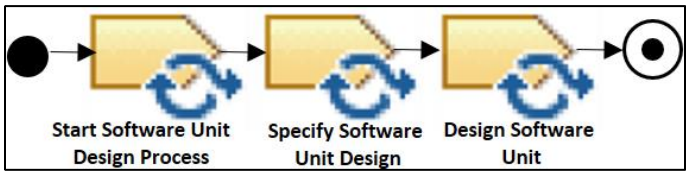

Start Software Unit Design Process
Specify Software Unit Design
Design Software Unit

Figure 95. Option for create the activity diagram

Once created, t the plugins are exported with the menu option "File → Export → XML → Export one or more method plugins" and select the three plugins modelled.

3.7.2 Guidelines for formalizing standard's requirements (*)

Two main activities are required for the formalization of standard requirements, as presented in Figure 96.

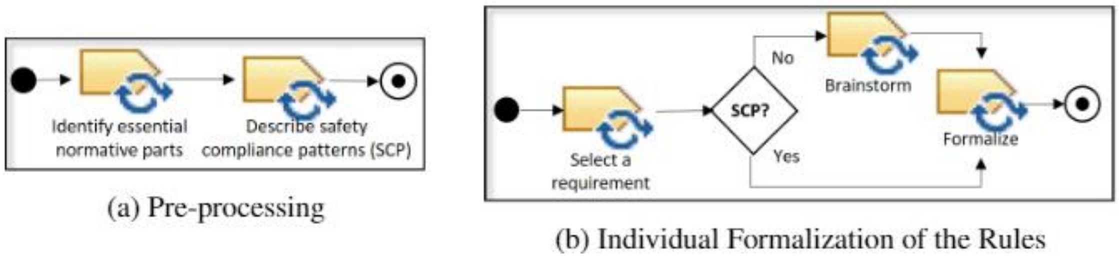

Identify essential normative parts
Describe safety compliance patterns (SCP)
(a) Pre-processing
No
Brainstorm
SCP?
Select a requirement
Yes
Formalize
(b) Individual Formalization of the Rules

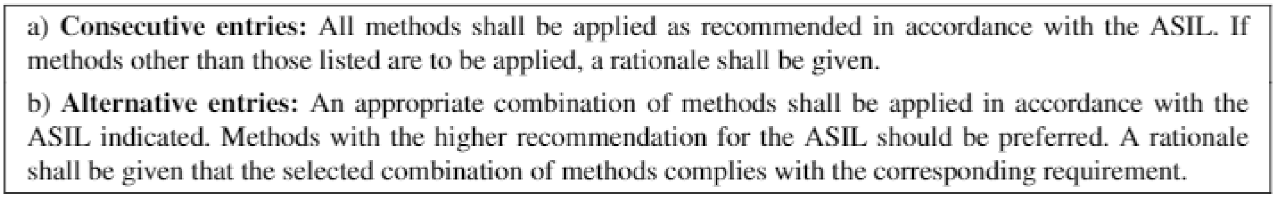
Figure 96. Methodological Guidelines

3.7.2.1 Pre-processing

The pre-processing is the suggested initial stage, which provides essential inputs for the formalization of standard's requirements. As the Figure 96 part a depicts, two tasks are required during this activity, i.e., the identification of the normative parts and the generation of safety compliance patterns. As an example, the preprocessing done with ISO 26262 [20] is presented.

1) Identify essential normative parts

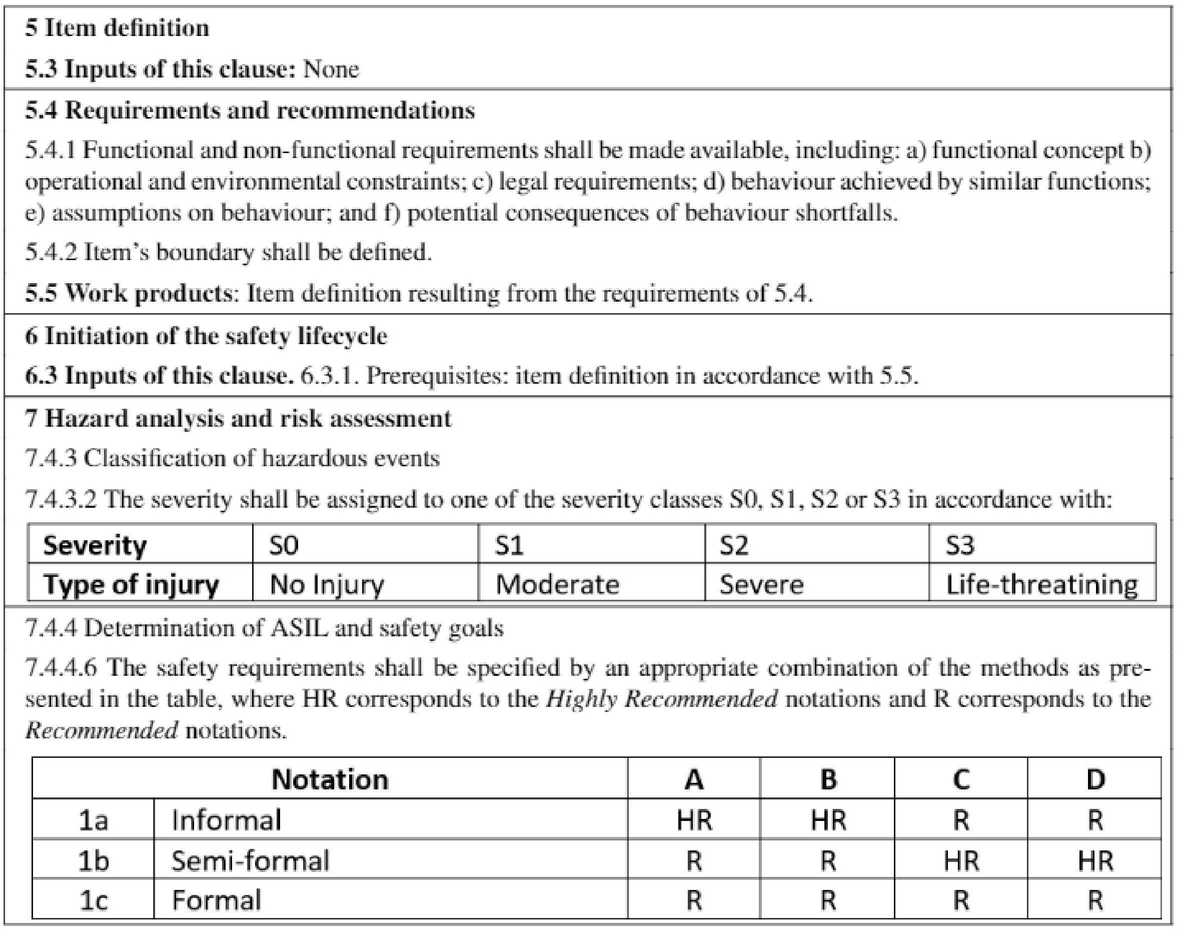
In this activity, the criteria for selecting the normative parts of ISO 26262 is presented. ISO 26262 has the following structure. It is composed by ten parts from which nine contain normative requirements. The normative parts are structured in a similar way, containing, a *foreword*, *introduction*, *bibliography*, annexes, and *clauses*. *Clause 1* recalls the general scope of the standard and situates the particular part in this scope. *Clause 2* recalls the normative references indispensable for the adoption of the specific part. *Clause 3* recalls the reference for terms, definitions, and abbreviated terms. *Clause 4* is of particular importance since it describes two compliance conditions required along all the standard. One of the conditions is related to the tailoring of the safety activities, which is valid if "*a rationale is available that the non-compliance is acceptable, and the rationale has been assessed*". The second condition is related to the interpretation of tables as is recalled in Figure 97.


a) **Consecutive entries:** All methods shall be applied as recommended in accordance with the ASIL. If methods other than those listed are to be applied, a rationale shall be given.

b) **Alternative entries:** An appropriate combination of methods shall be applied in accordance with the ASIL indicated. Methods with the higher recommendation for the ASIL should be preferred. A rationale shall be given that the selected combination of methods complies with the corresponding requirement.

Figure 97. Interpretation of Tables within ISO 26262:2011 (Clause 4.2)

Some of the remaining clauses represent phases of the safety process, which also describe activities and tasks. Each clause states its objectives, which describes the generic goals of the clause, general information, which gives an overall explanation of the clause, inputs of the clause, i.e., prerequisite and supporting information, requirements and recommendations, which describe the specific conditions that the process should fulfil, and finally the work products, which are the final elements that are to be generated. Notes and examples are expected to help the applicant in interpreting the requirements. The focus is on a subset of requirements from ISO 26262 part 3 presented in Figure 98.



5 Item definition

5.3 Inputs of this clause: None

5.4 Requirements and recommendations

5.4.1 Functional and non-functional requirements shall be made available, including: a) functional concept b) operational and environmental constraints; c) legal requirements; d) behaviour achieved by similar functions; e) assumptions on behaviour; and f) potential consequences of behaviour shortfalls.

5.4.2 Item's boundary shall be defined.

5.5 Work products: Item definition resulting from the requirements of 5.4.

6 Initiation of the safety lifecycle

6.3 Inputs of this clause. 6.3.1. Prerequisites: item definition in accordance with 5.5.

7 Hazard analysis and risk assessment

7.4.3 Classification of hazardous events

7.4.3.2 The severity shall be assigned to one of the severity classes S0, S1, S2 or S3 in accordance with:

Severity	S0	S1	S2	S3
Type of injury	No Injury	Moderate	Severe	Life-threatining

7.4.4 Determination of ASIL and safety goals

7.4.4.6 The safety requirements shall be specified by an appropriate combination of the methods as presented in the table, where HR corresponds to the *Highly Recommended* notations and R corresponds to the *Recommended* notations.

Notation		**A**	**B**	**C**	**D**
1a	Informal	HR	HR	R	R
1b	Semi-formal	R	R	HR	HR
1c	Formal	R	R	R	R

Figure 98. ISO 26262: 2011 part 3

Taking into account the structure of ISO 26262, the formalization process of each part starts from clause 5, which contains specific normative requirements. Within each clause, the information under the titles *objectives* and g*eneral*, as well as the *notes* and the *examples* embedded in the text, do not prescribe the process to be adopted. Therefore, this information is not formalized but it can be used to provide context and explanation for the application of the requirements. Instead, the information under the titles *prerequisites* and *work products* should be taken into consideration whenever a process is expected to be represented via a model embracing input/output elements. Also, *requirements and recommendation* contain the essential norms for constraining the breakdown and execution of the work as well as for guiding on how it should be planned and executed. The title of the clause may suggest the initiation of a phase in the life-cycle. Therefore, it should be also taken into account in the formalization process.

2) Describe safety compliance patterns

Some parts of ISO 26262 are described in a specific way, i.e., they are contained in specific structures that repeat in the whole text. These structures are considered to conform to safety compliance patterns that can be used as possible mechanisms to facilitate the formalization process within ISO 26262. The patterns are listed and described below.

- **Initiation of a phase**: The title of a clause may suggest the initiation of a phase within the safety process. Initiation of a phase is formalized as a constitutive rule. However, prerequisites may be needed. In this case, the prerequisite, which can be work products or previous clauses, must be in place before the phase starts, i.e., there is an obligation of having them. Within the prerequisites (if they exist), there is the obligation to initiate the clause.

- **Provision of a rationale**: The presentation of a rationale implies *compliance with conditions*. As recalled previously, the rationale for being valid must be assessed. Therefore, attaching a rationale implies the obligation of its verification by a domain expert.
- **Consecutive entries**: These methods are listed in tables with a sequence number in the leftmost column, e.g., 1, 2. The normative provision (See Figure 97, line a) is that after the ASIL is verified, there is an obligation to provide all listed methods. However, the provision of other methods, the attachment of a rationale supporting the application of other methods and the verification of the rationale by a domain expert grants a permit for not providing all the recommended methods.
- **Alternative entries**: There are methods in tables that are listed in an alternative way, i.e., marked by a number followed by a letter in the leftmost column, e.g., 2a, 2b. The normative provision (See Figure 97, line b) is that after the ASIL is verified, there is an obligation to provide a combination of the methods listed. The combination of these methods also obliges to include those marked with the highest recommendation level and requires a rationale explaining how the combination of methods complies with the requirement. However, the provision of other methods, the attachment of a strong rationale supporting the inclusions of other methods and the verification of the rationale by a domain expert, grant the permit for not including the highest recommended methods listed in the table.
- **Guidance**: ISO 26262 includes the provision of techniques and methods expressed in the form of guidance. In general, guidance is a procedure to build something else, and in this sense, the guidance elements become preconditions. To comply with the normative provision guidance, the procedure that is affected by the guidance is only obliged after all the elements of the guidance are provided.
- **Constitutive Tables**: In this kind of tables the intersection of two or more entries provides the information required for complying with a specific requirement. The normative provision is that after having the inputs of the table, the output is obliged.
- **Work Product:** The resulting work products are always defined in the standard within the requirements that generated them. Therefore, these requirements are presented as antecedents that obliged the provision of the related work product.

3.7.2.2 Individual formalization of rules

The process for individual formalization of rules is presented in Figure 96 part b. In the first step, the user needs to select a requirement from the standard. that the suggestion is that the user select the requirements in the order they are presented and that the rules are named in accordance to the requirement to ensure consistency and traceability. For example, if a requirement is marked with the number *5.1*, the rule is called *r.5.1*. If the requirement must be divided in several rules, the name of the rule should be named by a number that accompanies the requirements plus a letter, i.e., *r.5.1.a, r.5.1.b.* Then, the user checks if the requirement responds to the description of one of the safety compliance patterns presented above. If yes, the requirements could be formalized according to the description presented for the pattern. Otherwise, brainstorming between the participants of the formalization process may be required. Following it is presented, with an example, the formalization process.

Clause 5 of ISO 26262 part 3 determines that it is an obligation to initiate the phase *Item definition.* Therefore, as the pattern "*Initiation of a phase"* suggest, the formalization starts with the obligation in this case without prerequisites to perform item definition (see rule r5). Requirement 5.4.1. defines a guidance to provide the functional and non-functional requirements for the item definition. As it is described in the pattern "*Guidance"*, initially the components of the guidance should be provided, and then, they conform the element itself (see rules r5.4.1.a to r5.4.1.f and then r5.4.1). Requirement 5.4.2 prescribes the provision of the boundary of the item, which is an obligation that is the result of performing item definition (see rule r5.4.2). The last requirement of this clause, the one numbered as 5.5, defines the work product resulting from this phase. As the pattern “*Work Product”* describes, the requirements are presented as antecedents (specifically, clause 5.4) of the needed work product (see r.5.5). The rules are presented in Figure 99.

$$\mathbf{r5:} \Rightarrow [O]performItemDefinition$$
$$\mathbf{r5.4.1.a:} performItemDefinition \Rightarrow [O]provideFunctionalConcept$$
$$\mathbf{r5.4.1.b:} performItemDefinition \Rightarrow [O]provideOperationalEnvironmentalConstraints$$
$$\mathbf{r5.4.1.c:} performItemDefinition \Rightarrow [O]provideLegalRequirements$$
$$\mathbf{r5.4.1.d:} performItemDefinition \Rightarrow [O]provideBehaviorAchievedBySimilarFunctions$$
$$\mathbf{r5.4.1.e:} performItemDefinition \Rightarrow [O]provideAssumptionsOnBehavior$$
$$\mathbf{r5.4.1.f:} performItemDefinition \Rightarrow [O]providePotentialConsequencesOnbehaviorShortfalls$$
$$\mathbf{r5.4.1:} provideFunctionalConcept, provideOperationalEnvironmentalConstraints, provideLegalRequirements$$
$$provideBehaviorAchievedBySimilarFunctions, provideAssumptionsOnBehavior,$$
$$provideConsequencesBehaviorShortfalls \Rightarrow [O]provideFunctionalAndNonFunctionalRequirements$$
$$\mathbf{r5.4.2:} performItemDefinition \Rightarrow [O]defineItemBoundary$$
$$\mathbf{r.5.5:} provideFunctionalAndNonFunctionalRequirements, defineItemBoundary \Rightarrow [O]produceItemDefinition$$

Figure 99. Rule set defined for clause 5

To show how to formalize prerequisites, the requirement 6.3 is recalled. As the pattern “*Initiation of a phase”* suggest, the prerequisites, in this case the production of the *item definition* defines the obligation of performing the phase “*Initiation of the safety lifecycle”* (see rule r6.3). This rule is presented in Figure 100.

$$\mathbf{r6.3:} produceItemDefinition \Rightarrow [O]performInitiationSafetyLifeCycle$$

Figure 100. Rule set defined for clause 6

Clause 7.4.3 is formalized to show how to use the pattern called “*Constitutive Tables*”. As the pattern suggests, the intersection of the entries in the constitutive tables are the antecedents of the rules. The formal representation of this requirement is presented in Figure 101.

$$\mathbf{r7.4.4.a:} provideASILA \Rightarrow [O]provideCombinationOfTheListedMethods$$
$$\mathbf{r7.4.4.b:} provideCombinationOfTheListedMethods \Rightarrow [O]includeInformalNotations$$
$$\mathbf{r7.4.4.c:} includeInformalNotations \Rightarrow [O]attachWeakRationaleSupportingListedMethods$$
$$\mathbf{r7.4.4.d:} attachWeakRationaleSupportingMethods \Rightarrow [O]VerifyWeakRationaleByDomainExpert$$
$$\mathbf{r7.4.4.e:} includeNotListedMethods, atachStrongRationaleSupportingNotListedMethods,$$
$$verifyStrongRationaleByDomainExpert \Rightarrow [P] - provideCombinationOfTheListedMethods$$
$$r7.4.4.d > r7.4.4.a$$

Figure 101. Rule set defined for clause 7

3.8 Reuse Discovery

This section includes the main guidance for the technologies currently developed in AMASS for semantics-based reuse discovery, i.e. for the identification of reusable information based on their semantics. This is divided into two areas: how system artefacts can be represented and the supported architecture and operations.

3.8.1 Methodology to represent system artefacts

The methodology to represent system artefacts is based on SRL (Figure 102; System Representation Language), which is a resource shape that provides the schema to define the input/output interface for reuse operations. Any input content and output artefact must be expressed following the entities of SRL and creating an underlying graph that is used by a search engine to discover potential reusable assets.

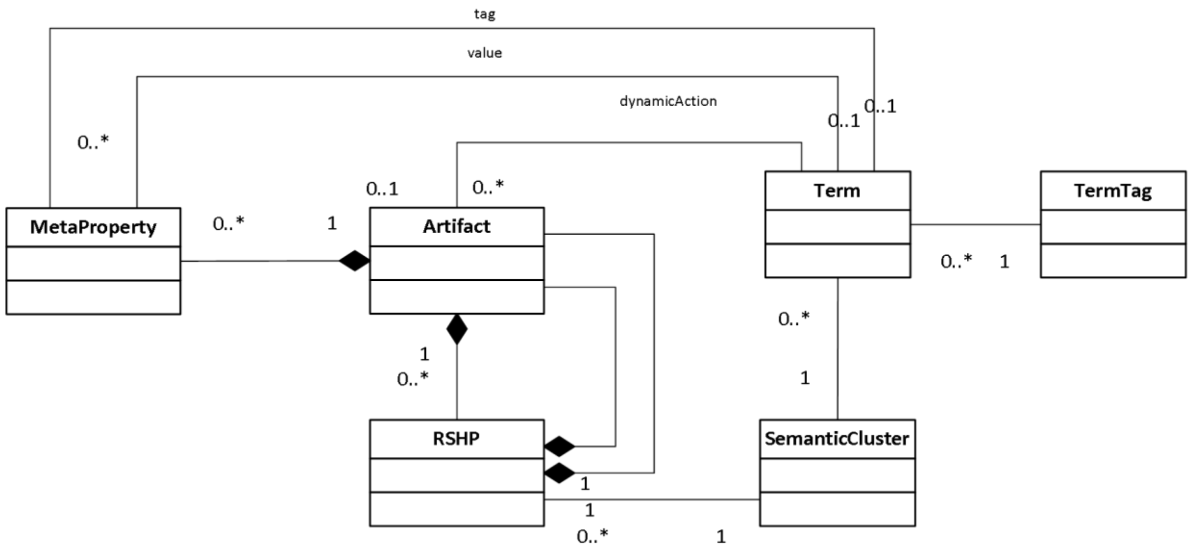


Figure 102. System Representation Language (SRL)

The definition of the entities of SRL and usage examples are as follows:

- *Artefact* is defined as a knowledge container and described by Relationships.
  Example: a diagram, a diagram node.
- *RSHP* (RelationSHiP) is the core component of SRL. It is used as a descriptor of all types of Artefacts information and to connect them.
  Example: to indicate that a diagram node is part of a diagram.
- *MetaProperty* represents metadata about an Artefact, typically through key and value pairs.
  Example: the version of a diagram.
- *Term* represents a semantic concept in a natural language.
  Example: “car”, “sensor”.
- *TermTag* represents the syntactic category of a term.
  Example: noun, verb.
- *Semantic Cluster* represents that semantic category (or type) of an element.
  Example: ‘system’ for “car”, “class” for a diagram node in a class diagram.

Further details about foundations for system artefact representation are presented in D6.3 [12]. Table 1 presents a generic mapping between elements of a diagram and SRL entities.

Table 1. Mapping between generic model concepts and SRL

Generic Object	SRL Entity
Model	Artefact
Relation	RSHP
Relation Type	Semantic Cluster
Node	Artefact
Node Property Attribute	Metaproperty

3.8.2 Operations to support reuse discovery

Table 2 presents the base operations necessary for reuse. CRUD operations correspond to Create, Read, Update, and Delete. They are introduced in D6.3 [12]. The application of these operations enable semantics-based reuse of the information stored in ontologies, leading to what could be regarded as configuration management of ontologies (see Figure 103; further explained below through specific operations).

Table 2. Operations for the management of the System Knowledge Base and the System Assets Store

	Core Operations				Common Operations				Retrieval Operations	Reuse Operations		
Operations for the management of the System Knowledge Base												
Resource	**C**	**R**	**U**	**D**	**Text Standardization**	**Validation**	**Preferred**	**Related**	**Index & Search**	**CopyTo**	**Merge**	**Diff**
Term	x	x	x	x	x	x	x	X	x	x	x	
SCM (Relationships)	x	x		x	x	x			x	x	x	
Pattern	x	x		x					x	x	x	
Rule	x	x		x					x	x	x	
Operations for the management of the System Assets Store												
Artefact	x	x	x	x	x	x	x	X	x	x	x	x

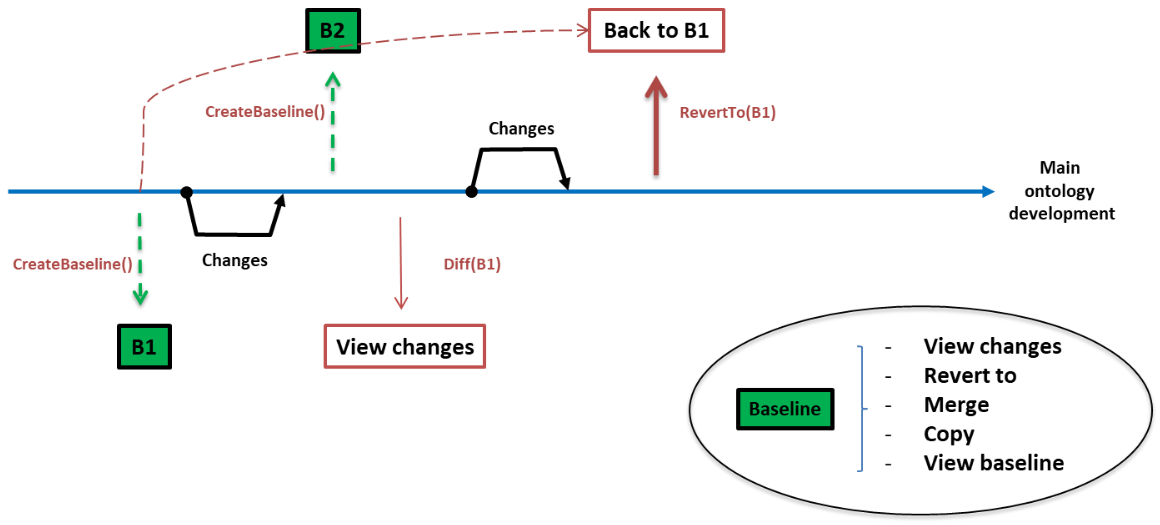
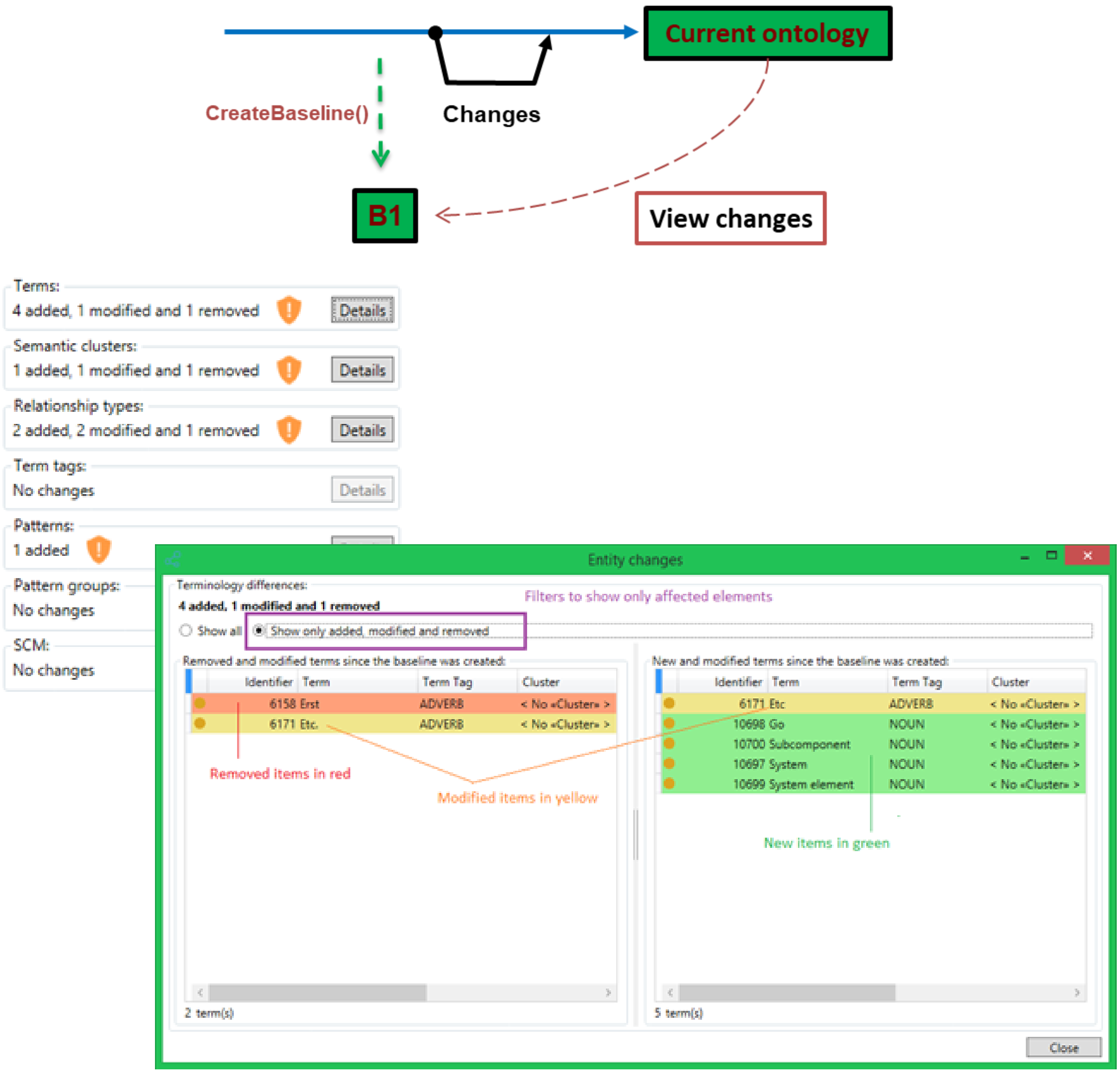


Figure 103. Ontology configuration management: overall workflow

When evolving an ontology, one might wonder what the evolution of the ontology itself has been during its lifecycle. In addition to that, if a mistake is found, it would be helpful to revert the ontology to one of the previous states. For these reasons, some operations have been implemented to deal with ontology configuration management.

- **View changes** (Figure 104) allows the user to graphically visualize differences between the current ontology and one of its ontology baselines (or against another ontology). It shows the new, modified and removed elements.



Terms:
4 added, 1 modified and 1 removed Details

Semantic clusters:
1 added, 1 modified and 1 removed Details

Relationship types:
2 added, 2 modified and 1 removed Details

Term tags:
No changes Details

Patterns:
1 added

Pattern groups:
No changes

SCM:
No changes

Terminology differences:
4 added, 1 modified and 1 removed

Show all [✓ Show only added, modified and removed]

Removed and modified terms since the baseline was created:

Identifier	Term	Term Tag	Cluster
6158	Erst	ADVERB	< No «Cluster» >
6171	Etc.	ADVERB	< No «Cluster» >

2 term(s)

New and modified terms since the baseline was created:

Identifier	Term	Term Tag	Cluster
6171	Etc	ADVERB	< No «Cluster» >
10698	Go	NOUN	< No «Cluster» >
10700	Subcomponent	NOUN	< No «Cluster» >
10697	System	NOUN	< No «Cluster» >
10699	System element	NOUN	< No «Cluster» >

5 term(s)

Figure 104. Ontology configuration management: "View changes" scenario

- **Copy** (Figure 105) allows the user to copy content to another ontology, considering all the dependencies. The user can decide either a total copy (the whole ontology) or a partial copy (by manually selecting the elements to copy).

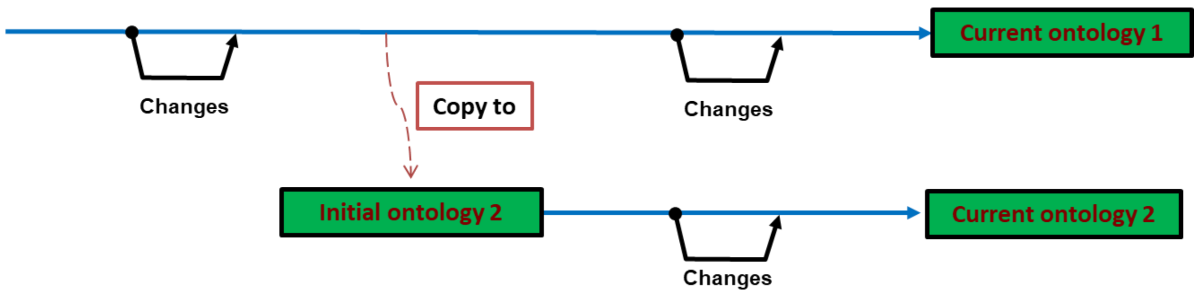


Figure 105. Ontology configuration management: “Copy” scenario

- **Merge** (Figure 106) allows the user to merge the content of an ontology with the content of one of its baselines or another ontology.

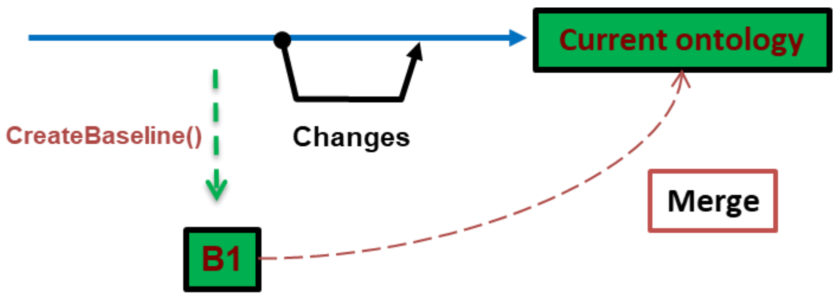


Figure 106. Ontology configuration management: “Merge” scenario

- **Revert** (Figure 107) allows the user to revert the current ontology to one of the previous states.

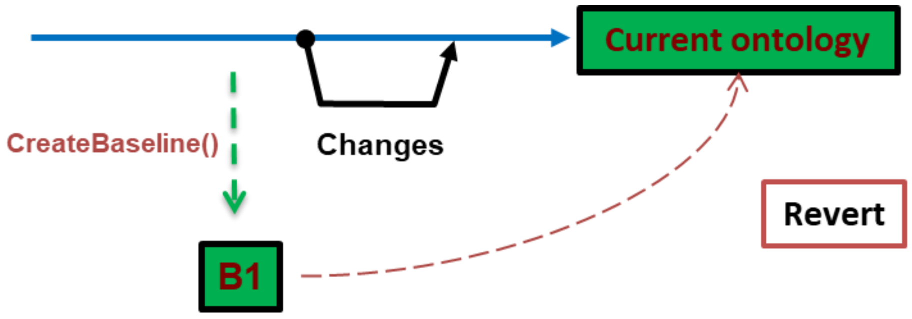


Figure 107. Ontology configuration management: ‘Revert” scenario

All these reuse operations have been implemented in Knowledge Manager (Figure 108).

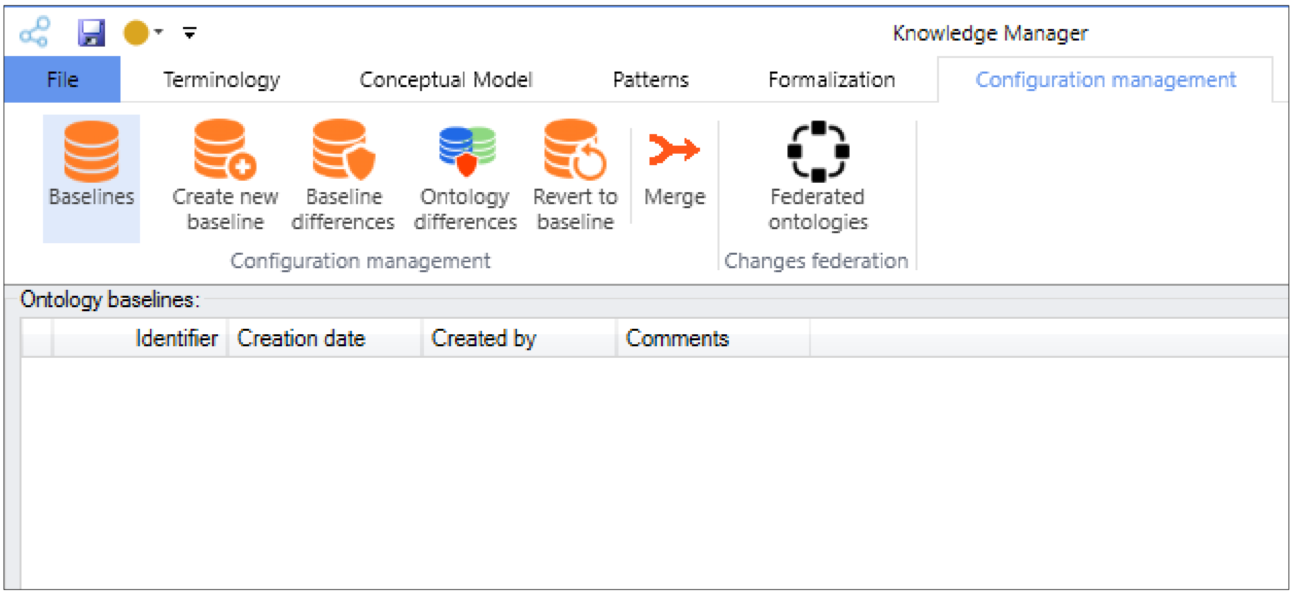


Figure 108. Ontology configuration management in Knowledge Manager

3.9 Representation of Safety Standards with Semantic Technologies (*)

The approach to represent safety standards with the semantic technologies provided by Knowledge Manager (KM) is presented in D6.3 [12]. The activity diagram in Figure 18 synthesises the approach:

- KM configuration. This activity is necessary to tailor the default KM usage to represent safety standards, i.e. certain aspects of KM must be configured so that a user can create a suitable representation in accordance with the holistic generic metamodel. The configuration focuses on those semantic aspects of the standards that must be included in the representation. These aspects are specific to safety standards but independent of the specific standard to represent.
- Specification of a standard's information. This activity results in the specific representation of a given safety standard. Two tasks can be distinguished. These tasks will usually be executed iteratively to incrementally represent a safety standard.

As additional guidance, Table 3 presents the mapping between the concepts of a Reference Assurance Framework (Figure 18) and the concepts managed by Knowledge Manager, and some examples.

Within the overall purpose of demonstrating alignment or compliance with a safety standard, the situations in which it would be possible to take advantage of the representations include:

a) Quality analysis of the text of a safety standard. KM is part of a tool suite that supports, among other features, system artefact quality analysis, including textual artefacts. More concretely, the suite can analyse artefact correctness, completeness, and consistency. Considering the text of a safety standard as an example of artefact, its text quality could be determined. This would be valuable because text quality is one of the most frequent weaknesses that practitioners find in safety standards. Parts that could be better specified or should be clarified could be identified.

b) System specification alignment. When specifying information for a specific system or analysing the information, the degree to which the specification is aligned with a given standard could be assessed. First, the system could be specified, e.g. its system requirement, according to patterns that refer to the semantic clusters added or to standard-specific terms. Second, an ontology of the system could be linked to the ontology of the standard, e.g. to specify that a given part of the system corresponds to the DO-178C component concept.

c) Compliance assessment. An ontology of a safety standard created with KM could be used to assess process and product compliance. The tool suite capabilities could be used to compare process or product information with the ontology, in order to determine compliance gaps. The information could correspond to artefacts of different nature: textual specifications, documents, diagrams, spreadsheets.
d) Comparison of standards. The text or ontology of a safety standard could be compared with the ontology of another standard, in order to identify commonalities and differences. This usage can be regarded as an extension of (a).
e) Reuse of compliant system information. If a system's information (e.g. a system model) is linked with the ontology of a safety standard to declare compliance with the standard, it would be possible to search for compliant system information and, when found, to reuse it. It could even be possible to analyse system information reuse between safety standards if the ontologies of the different standards are linked. The linking of a system's information with the ontology could be based on (b).
f) Specification of standard-specific metrics. Specific metrics could be designed within the Inference rules layer based on the semantic information of a safety standard represented in KM. The metrics could assess (1) general compliance with the standard (e.g. the amount of Reference Artefacts that have been provided) and (2) artefact-specific characteristics that a standard defines (e.g. architecture specification consistency). Although the metrics are often not directly declared in the safety standard (e.g. for the latter example), the standards' information would drive their definition by indicating the areas for which metrics could be designed and possible aspects to consider.

Table 3. Mapping between the concepts of a Reference Assurance Framework and the concepts managed by KM

Reference Framework Concept	KM Concept	Example/Comment
Reference Artefact	Cluster	Software requirements data (DO-178C)
Reference Activity	Cluster	Component design (EN 50128)
Reference Technique	Cluster	Formal methods (IEC 61508)
Reference Role	Cluster	Designer (ISO 26262)
Reference Requirement	-	Reference Requirements are not represented as elements of an ontology
Reference Artefact Attribute	Cluster	Rationale (for Derived High-Level Requirements; DO-178C)
Reference Artefact Relationship	Relationship type	Derived from (e.g. Software system design specification derived from software architecture design; IEC 61508)
Reference Criticality Kind	Cluster	ASIL (ISO 26262)
Reference Applicability Kind	Cluster	Objective satisfaction (DO-178C)
Reference Criticality Level	Cluster	ASIL A-D (ISO 26262)
Reference Applicability Level	Cluster	Objective satisfaction should be shown as Satisfied or Satisfied with independence, or is at applicant's discretion (DO-178C)
Reference Applicability	Relationship type	A cell in recommendation table in EN 50128 (which links several elements)
Associations between Reference Assurance Framework classes	Relationship type	User – Input Artefact

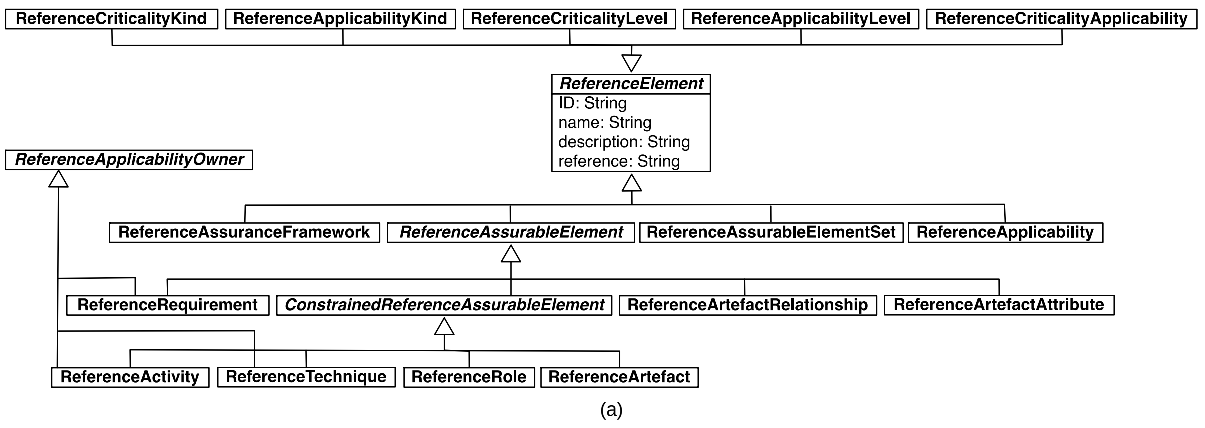
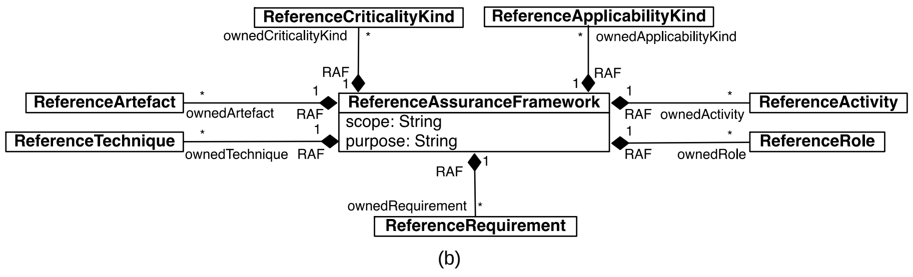
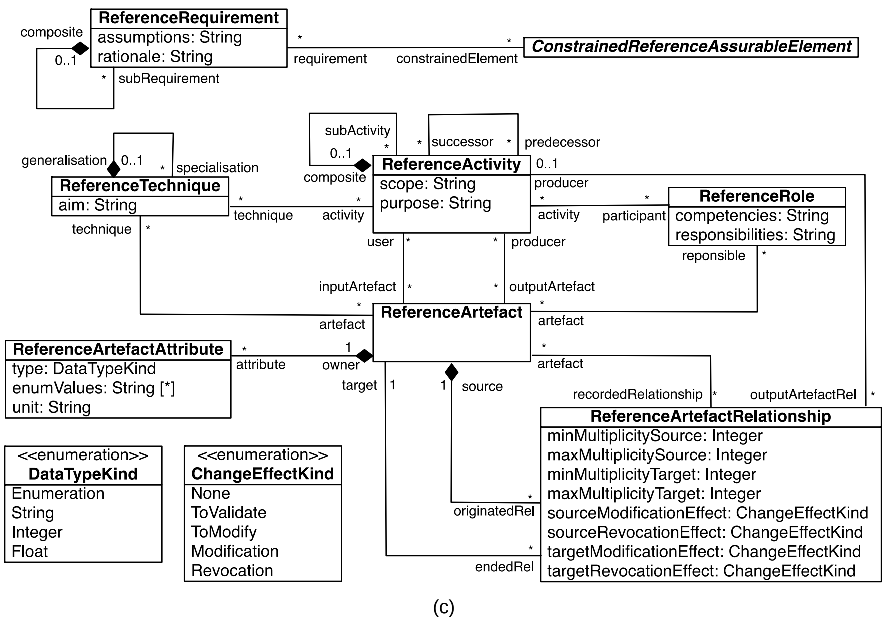
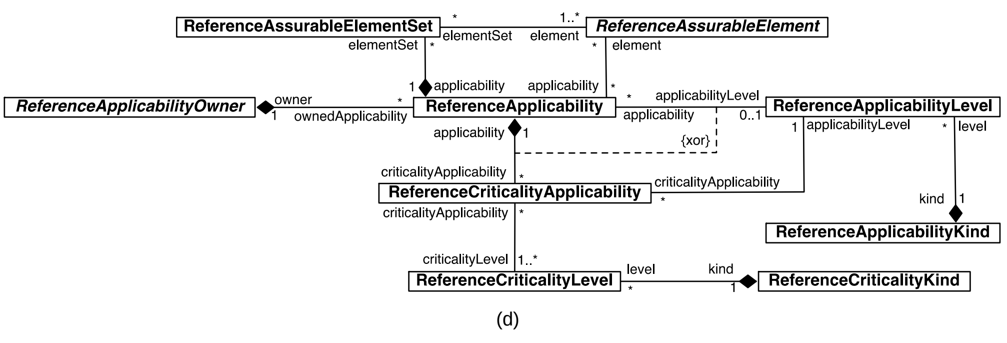





Figure 109. Reference Assurance Framework metamodel divided into (a) specialization hierarchy, (b) main components of a RAF, (c) reference assurable elements associations, and (d) RAF applicability information [18]

3.10 Methodology to Reuse Safety and Security Analysis Artefacts (*)

The Safety Architect tool is an external tool, which interacts with AMASS to provide a support for safety and security co-analysis from CHESS model. However, when a system model has progressed/modified, it can demand a huge amount of time to be analysed again. The EMF Diff/Merge[2] methodology can reduce this (re-)analysis cost.

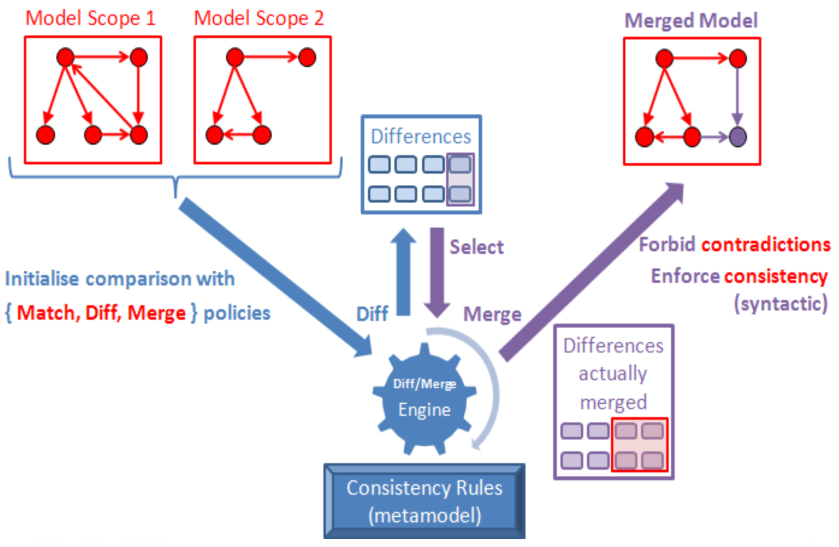
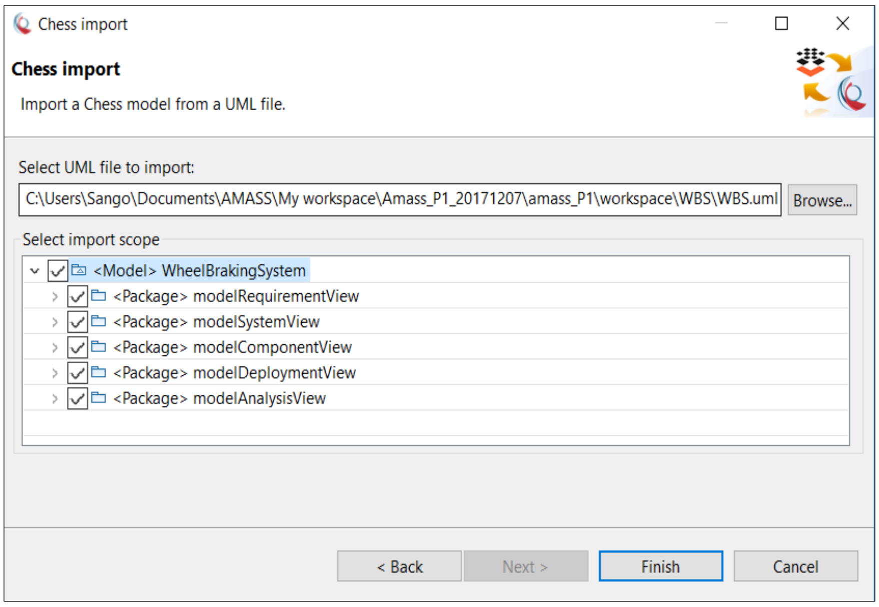


Figure 110. EMF Diff/Merge Method

The user can compare and merge two versions of the same model in order to reuse specifics artefacts, such as safety or security analysis artefacts contained in the version already analysed. To apply this Diff/Merge methodology supported by the interface between CHESS and Safety Architect, the user must follow the following steps:

- **Step 1:** Import a CHESS model from a UML file.



Chess import

Chess import

Import a Chess model from a UML file.

Select UML file to import:

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

Select import scope

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

Figure 111. Import from CHESS tool to Safety Architect tool

- **Step 2:** Perform Safety or Security Analysis in Safety Architect on the model.

2 http://wiki.eclipse.org/EMF_DiffMerge

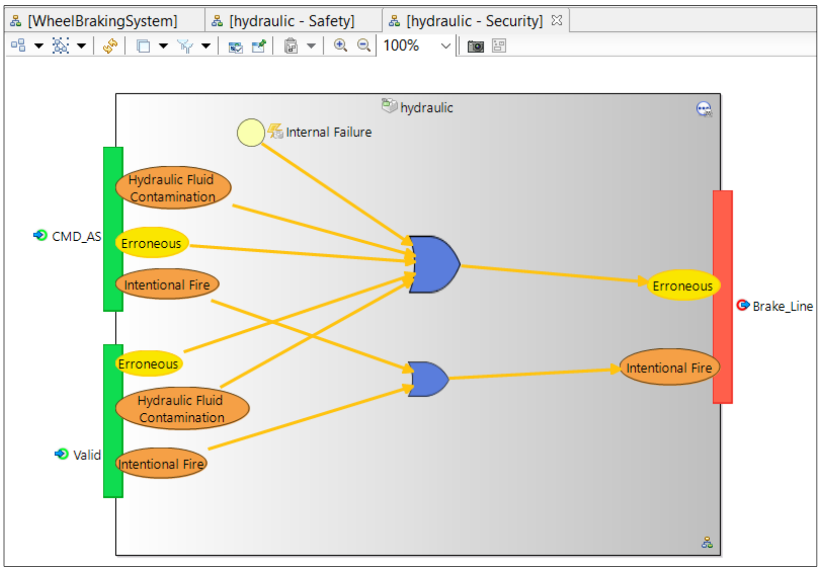


Figure 112. Safety & Security Viewpoint in Safety Architect tool

- **Step 3:** When the same CHESS model is modified, **re-import** the model by applying this Diff/Merger**.**

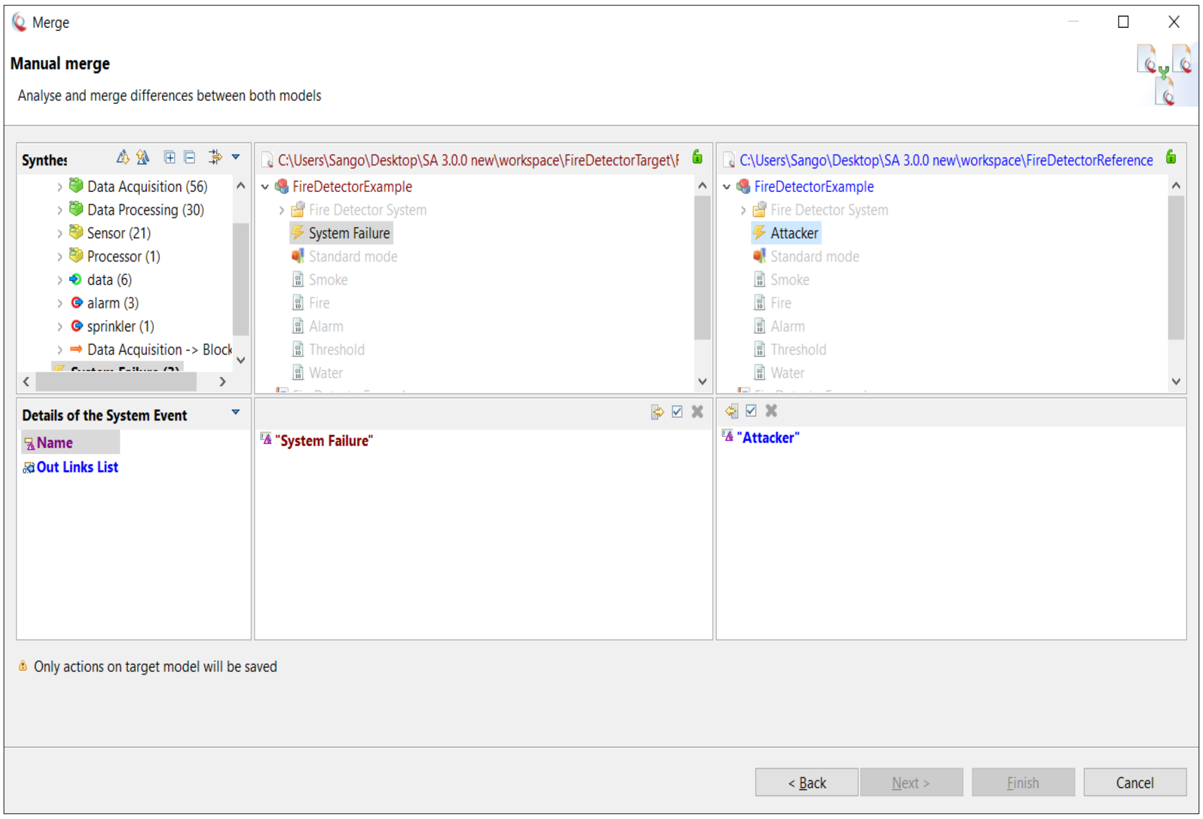


Figure 113. Differences between two models

To summarize, the Diff/Merge reuse mechanism can be exploited during the **re-import** from the AMASS Platform - CHESS tool to the Safety Architect tool. Indeed, if there is at least one existing project (imported from the CHESS) into the SA workspace, the user can apply the merge operation during the **re-import** of the same model. The usage scenario is as follows:

- Import a CHESS model into Safety Architect.
- Perform Safety or Security Analysis in Safety Architect on the model.
- When the same CHESS model is modified, re-Import the model by applying this Diff/Merger.

3.11 Model-Based Testing Methodologies (*)

This section includes a high-level description of the main ideas for how analysis and test may be combined in a common methodology consisting of common verification planning and verification status tracking, and general guidance for exploiting analysis and test results in subsequent verification steps is given.

The section introduces and describes an approach that allows to refine the overall methodology into dedicated solution guidelines by means of workflow patterns. This approach allows describing best practices solutions in a harmonized way; it also enables to bring the individual workflows into relation with each other. Therefore, it provides a means to refine the overall Model Based Analysis and Test (MBAT) methodology to individual solutions and their applications.

The proposed refinement consists of 4 levels, the overall method, combination workflow patterns, method instances. Figure 114 depicts the refinement process.

In general, a pattern is defined in the POSA book series [57]:

> “A (solution) pattern is a (generic) solution for a certain problem in a given context”.

This it is not a finished solution that can be transformed directly into a running system, but it is a principal description or template for how to solve a problem. A good pattern can be used in many different situations and defines a formalized best practice.

An analysis and test (A&T) pattern is a typical workflow executed during V&V that involves a mixture of verification techniques (dynamic testing/simulation and static code or model analysis). A pattern defines a sequence of the main processing steps of a V&V activity, not related to a specific tool. A step is typically automated, where tool support is required and feasible, but may be performed manually, in lack of tool support, but may take much effort. A step may also be manual where tool automation is expected to be hard, relies on informal information, or requires human “intelligence”. Also, it defines the kinds of data exchanged. Again, the logical content is emphasized over specific data formats (use metamodel concepts where applicable).

A method instance is an instantiation of a workflow pattern with specific tools, and specific types of data for analysis or test results, and model notations. Such a method is to be executed in the context of the MBAT RTP [55] providing a set of basic services, and the facilities for controlling execution and exchange of the defined artefacts. An A&T pattern must be instantiated by specific tools and defined data exchange formats. To do so efficiently they will make use of the MBAT RTP. This may also include a script language for automating V&V workflows.

The methodology consists of an overall workflow for combining model-based analysis and testing. The main idea is to use joint V&V planning to track verification status and based on an inspection of the requirements and V&V objectives, apply the best technique for that, and systematically asses the outcome for a subsequent refined iteration. Also, it develops the main V&V flow across most abstraction levels and indicates how the overall workflow may be applied across V&V activities and abstraction levels. To capture the commonalities of different verification techniques, the method defines the concept of a generic A&T step.

To make the A&T combination idea more concrete and operational, concrete ways of combining and exploiting combined A&T must be identified. The methodology defines a refinement process based on A&T patterns. An A&T pattern is a typical workflow tailored towards a specific purpose, and that involves a mixture of verification techniques (dynamic testing/simulation and static code or model analysis). A pattern defines sequence of the main processing steps of a V&V activity, not related to a specific tool. A method instance defines specific tools, exchanged documents and artefacts used to implement a pattern.

3.11.1 Introduction to Model-based Testing (MBT) with Patterns

Use models to guide and support testing of a target system (SUT – System Under Test). Increase confidence in generated tests by analysing (model-checking) the model from which tests are generated (valid tests that satisfy requirements, failing tests likely caused by wrong implementation, etc.). The overall workflow is described in Figure 114.

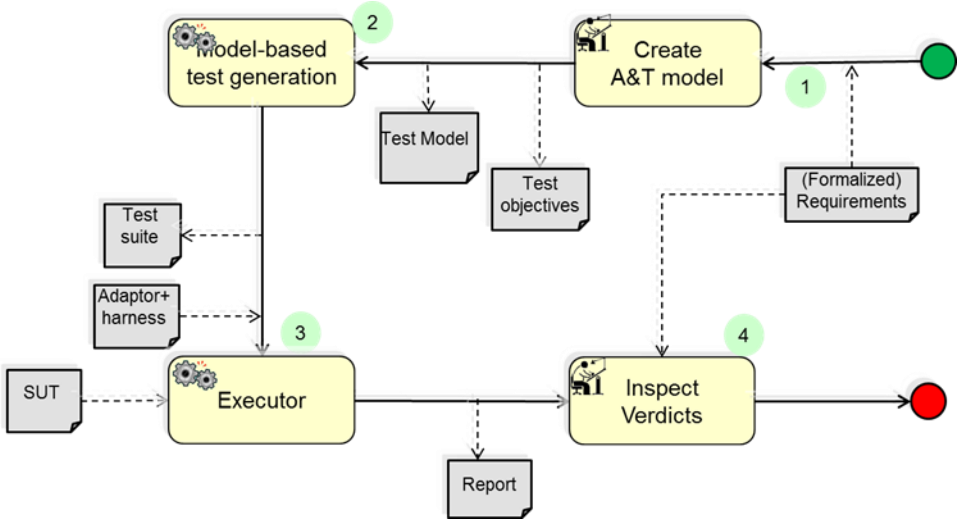


Figure 114. Structure of MBT (Model-Based Testing)

Participants:

- User: Person who provides the requirements; (s)he may also receive the reports.
- A&T Model Engineer: Person who cooperates with the user to analyse the requirements and creates the A&T (test) model that represents them in a formalised way.
- Test Objectives denote what shall be tested; they are closely related with the concept of coverage. For MBT, the most important coverage criterion is "each requirement shall be tested at least once". So, test objectives often represent requirements, but there may be an m:n relationship between requirements and test objectives.
- Test Suite: set of test cases generated (TCG) from test model. Each test case contains at least input data, but for assessing whether the resulting behaviour conforms to requirements, expected results should also be present. If the latter cannot be expressed by explicit values, conditions such as „x > y" are used. Conditions about results are also called oracle.
- SUT: The system to be tested.
- Test Executor: Unit that applies test cases to the SUT and compares results with actual ones (i.e. executes the "verdict"). Usually, a so-called "test harness" and "adaptor" is used, a test environment that feeds input data of a test case to the SUT and records outputs and behaviour of the SUT, e.g. by mocking services used by the SUT. Often, test experts cooperate with SUT experts in operating the test execution.

Actions/Collaborations:

1. The A&T model engineer cooperates with the user during development of the test model, in order to understand the requirements and model them correctly.
2. The test engineer uses the test model to generate test cases from it automatically. This may require feedback to the model engineer in the case of problems, e.g. that the requested coverage cannot be achieved, or the model is too complex.

3. The test engineer cooperates with SUT experts (the user) in operating the test execution environment. The test environment usually must be adapted to the SUT, for which knowledge about the SUT is needed. Sometimes, also adaptations of the SUT must be done, e.g. for logging and reporting.
4. Taking oracles and test results, the verdict expresses whether a test passed (the test results fulfil the oracle), failed (this is not the case), or is considered as inconclusive. The latter occurs when the SUT reacts in a way not foreseen by the model such that the oracle cannot be applied. As there may be several reasons for that:
    - the model doesn't reflect the specific SUT's behaviour
    - the test case generation doesn't consider the alternative the SUT has taken
    - the report is not detailed enough.

   At least on inconclusive test cases, test experts need to interact with other experts to clarify this situation. On failed test cases, cooperation between involved persons is needed to identify and presumably to remove the reason.

3.11.2 Pattern for MBA-Based Runtime Verification

This is also an architectural consistency checking. The purpose is to ensure that the runtime behaviour of the system matches the models that are used as the input for static model-based analysis. In performing model-based analysis, some assumptions are considered (e.g. Worst-Case Execution Time WCET). By testing a system with respect to these assumptions, it is verified whether they are valid at runtime or not. This is of great importance because any inconsistency between the models capturing the expected behaviour and the actual behaviour of the system at runtime imply that the result of model-based analysis may not be valid and applicable anymore for the implemented system.

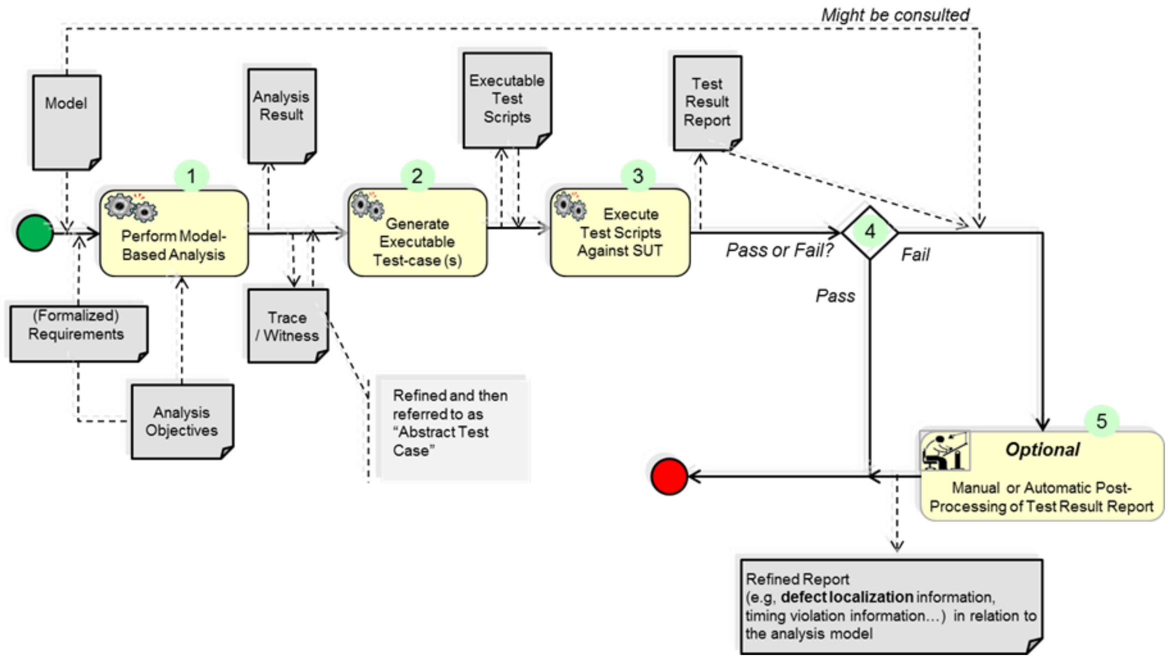

Might be consulted
Model
Analysis Result
Executable Test Scripts
Test Result Report
1
Perform Model-Based Analysis
2
Generate Executable Test-case (s)
3
Execute Test Scripts Against SUT
Pass or Fail?
4
Fail
Pass
(Formalized) Requirements
Trace / Witness
Analysis Objectives
Refined and then referred to as "Abstract Test Case"
5
Optional
Manual or Automatic Post-Processing of Test Result Report
Refined Report (e.g. defect localization information, timing violation information…) in relation to the analysis model

Figure 115. Structure of Model-Based Run-Time Behaviour Verification

Participants: only those are described here that operate in addition to those described in the pattern shown in Figure 115:

- <u>Analysis objectives</u>: properties that must be possessed by the test model. They are derived from requirements by the analysis and test (A&T) model engineer and may be formalized, e.g. as Time Constraint Tree Logic (TCTL) properties.

Actions/Collaborations:

1. Using the test objectives, the system model is analysed, resulting in traces through the model that represent witness for fulfilment the requirements. These traces represent test cases, but in abstract notation, because they cannot be directly applied to the SUT.
2. The test engineer turns these abstract test cases (automatically) into *executable test* cases, typically in form of test scripts, which can be understood by the test environment. This is mostly achieved by a rather straightforward notation transformation.
3. The test engineer executes the test cases against the SUT and records the test outcomes in the test reports.
4. The test engineer evaluates whether tests are passed of failed, typically by comparing reports with the intended results (test oracles).
5. For failed tests it can be appropriate to post-process the test report manually by the test engineer, to provide additional information such as suspected fault location, which were gathered during the test result evaluation.

3.12 Summary (*)

The following Table 4 summarizes the supported activities during specific development phases.

Table 4. Summary of reuse activities during specific development phases.

Phase	Supported activities
1. Digitalization of standards	Modeling standard's rules (requirements, activities, roles, applicability tables, criticalities…) can be done using OpenCert. Modeling of the standard's requirements and the rules in EPF Composer can be used to support the process planning phase.
2. Reuse preparation	Define reusable process elements in EPF-Composer to be used during the project. Define the equivalence mappings between the standards using OpenCert. Representation of Safety Standards with Semantic Technologies (one of its possible applications) using OSLC-KM
3. Assurance Project creation	Use the reuse discovery functionality (either using elastic search or OSLC-KM) in order to use the cross-project reuse in OpenCert. Use the reuse assistance to take advantage of the cross-standard reuse in OpenCert
4. Assurance case edition	Within EPF-Composer connect to OpenCert to use Automatic generation of process-based argumentation. From your system architecture design in Chess connect to OpenCert and automatically generate product-based argumentation. Manage the variability in the assurance case with BVR. Reuse of arguments using the argument patterns and modules in the assurance case editor provided in OpenCert.
5. Validation and verification activities	Reuse of safety and security analysis artefacts using Safety architecture and take advantage of model-based testing
6. Standards compliance	At any time, the user could either use the mappings table to check the status of compliance or from EPF-Composer and in combination with Regorous to check the compliance via the use of Formal Contract Logic.

4. Conclusions

This document defines a methodological guide to use the AMASS Platform for a cross-domain and intra-domain reuse approach with the AMASS Tool Platform and related external tools integrated with the platform. The deliverable at hand describes workflows and steps to perform the activities.

This version of the guide is related to the third and final prototype of the AMASS Tool Platform, called P2. It provides a variety of techniques to reuse different types of intra- and cross- domain assurance information. This information may be process-, product- or assurance case-related. This deliverable provides detailed guidance about reuse assistance, management of lines, automatic generation of arguments, transformation of standards to baseline models, compliance mapping, compliance checking, reuse discovery, representation of safety standards with semantic technologies, reuse of safety and security artefacts and model-based testing.

Abbreviations and Definitions

Abbreviation	*Explanation*
A&T	Analysis and Test
AIT	Assembly Integration and Test
API	Application Programming Interface
ASIL	Automotive Safety Integrity Level
AOCS	Attitude and Orbit Control Subsystem
BCL	Basic Constraint Language
BVR	Base Variability Resolution - a domain-specific language designed specifically to enable software product-line engineering (SPLE)
BVR-T	Base Variability Resolution Tool
CACM	Common Assurance and Certification Metamodel
CAN	Controller Area Network
CBSE	Component-Based Software Engineering
CCU	Central Computing Unit
CDO	Connected Data Objects
CHESSML	CHESS Modelling Language
COTS	Commercial Off-The-Shelf
CPS	Cyber-Physical Systems
CRUD	Create, Read, Update, and Delete
CVL	Common Variability Language
ECSEL	Electronic Components and Systems for European Leadership
ECSS	European Cooperation for Space Standardization
ECU	Electronic Control Unit
EMC^2	Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments
EMF	Eclipse Modelling Framework
EN	European Norm
EPF-C	Eclipse Process Framework-Composer
FBS	Functional Breakdown Structure
FCL	Formal Contract Logic
FMEA	Failure Modes and Effects Analysis
FMVEA	Failure Modes, Vulnerabilities and Effect Analysis
FODA	Feature-Oriented Domain Analysis
FTA	Fault Tree Analysis
GSN	Goal Structured Notation
GUID	Globally Unique Identifier
HTTP	Hypertext Transfer Protocol
HW	Hardware
ICSE	International Conference on Software Engineering
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
ISSRE	International Symposium on Software Reliability Engineering

JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JU	Joint Undertaking
KM	Knowledge Manager
LIN	Local Interconnect Network
MARTE	Modelling and Analysis of Real Time and Embedded systems
MBA	Model-Based design methodology for Assessing performance and safety requirements of critical systems
MBAT	Model Based Analysis and Test
MBT	Model-Based Testing
MCU	Microcontroller
MOF	Meta-Object Facility
NL	Natural Language
OMG	Object Management Group
OPENCOSS	Open Platform for EvolutioNary Certification of Safety-critical Systems
OSLC	Open Services for Lifecycle Collaboration
PBS	Product Breakdown Structure
PCB	Printed Circuit Board
PLE	Product-line Engineering
RAF	Reference Assurance Framework
RecL	Recommendation Level
REST	Representational State Transfer
RSHP	Relation SHiP, (Information representation model based on relationships)
RTP	Reference Technology Platform
SACM	Structured Assurance Case Metamodel
SAE	Society of Automotive Engineers
SCM	System Conceptual Model
SecRL	Security Risk Level
SEooC	Safety Element out of Context
SIL	Safety Integrity Level
SiSoPLE	Security-informed Safety-oriented Process Line Engineering
SoPLE	Safety-oriented Process Line Engineering
SPEM	Software & Systems Process Engineering Metamodel
SRL	System Representation Language
STO	Scientific Technical Objective
SUT	System Under Test
SVN	SubVersion
SysML	System Modelling Language
TCTL	Time Computation Tree Logic
TCG	Test Cases Generated
TTM	Time To Market
UMA	Unified Method Architecture
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

V&V	Verification and Validation
WEFACT	Workflow Engine for Analysis, Certification and Test
WCET	Worst-Case Execution Time
WP	Work Package
XMI	XML Metadata Interchange
XML	EXtensible Markup Language

References

[1] AMASS Deliverable D1.5 AMASS demonstrators (b), April 2018
[2] AMASS Deliverable D2.5 AMASS user guidance and methodological framework, November 2018
[3] AMASS Deliverable D3.3 Design of the AMASS tools and methods for architecture-driven assurance (b), March 2018
[4] AMASS Deliverable D3.7 Methodological guide for architecture-driven assurance (b), November 2017
[5] AMASS Deliverable D3.8 Methodological Guide for Architecture-Driven Assurance (b), October 2018
[6] AMASS Deliverable D4.8 Methodological guide for multiconcern assurance (b), October 2018
[7] AMASS Deliverable D5.8 Methodological guide for seamless interoperability (b), October 2018
[8] AMASS Deliverable D4.3 Design of the AMASS tools and methods for multi-concern assurance (b), April 2018
[9] AMASS Deliverable D4.7 Methodological guidance for multi-concern assurance (a), December 2017
[10] AMASS Deliverable D5.7 Methodological guide for seamless interoperability (a), December 2017
[11] AMASS Deliverable D6.1 Baseline and requirements for cross/intra-domain reuse, September 2016
[12] AMASS Deliverable D6.3 Design of the AMASS tools and methods for cross/intra-domain reuse, July 2018
[13] AMASS Deliverable D6.6 Implementation for cross-domain and intra-domain reuse, October 2018
[14] AMASS Deliverable D6.7 Methodological guide for cross/intra-domain reuse (a), January 2018
[15] AMASS Deliverable D5.3 Design of the AMASS tools and methods for seamless interoperability (b), June 2018.
[16] The OPENCOSS project (Open Platform for EvolutioNary Certification Of Safety-critical Systems) http://www.opencoss-project.eu/
[17] The SafeCer project (Certification of Software-intensive Systems with Reusable Components) http://cordis.europa.eu/project/rcn/103721_en.html and http://cordis.europa.eu/project/rcn/105610_en.html
[18] de la Vara, J.L., Ruiz, A., Attwood, K., Espinoza, H., Panesar-Walawege, R.K., Lopez, A., del Rio, I., Kelly, T.: Model-Based Specification of Safety Compliance Needs: A Holistic Generic Metamodel. Information and Software Technology 72: 16-30 (2016)
[19] The REUSE Company: Knowledge Manager. Online, https://www.reusecompany.com/knowledgemanager (2017)
[20] ISO 26262: "Road Vehicles-Functional Safety. International Standard," 2011
[21] EN 50128 Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems. CENELEC - Comité Européen de Normalisation Électrotechnique, June 2011
[22] DO 178C: Radio Technical Commission for Aeronautics (RTCA). Software Considerations in Airborne Systems and Equipment Certification, 2012
[23] SAE J3061: "Surface Vehicle Recommended Practice," Tech. Rep., 2016
[24] Eclipse process framework project (EPF), https://eclipse.org/epf/, accessed: 2017-09-12
[25] EPF Composer Architecture Overview. https://www.eclipse.org/epf/composer_architecture/
[26] The CHESS tool, https://www.polarsys.org/projects/polarsys.chess
[27] European cooperation for space standardization, ecss-e-st-40c, space engineering software. http://wwwis.win.tue.nl/2R690/doc/ECSS-E-ST-40C(6March2009).pdf. Last accessed: June 16, 2017
[28] Object Management Group (OMG). 2004. Software Process Engineering Metamodel Specification (SPEM), Version 1.1. ftp://ftp.omg.org/pub/spem-rtf/SPEM-CD-20040308.pdf. (2004). (Last accessed: November 21, 2017)

[29] Object Management Group (OMG). 2008. Software & Systems Process Engineering Metamodel Specification (SPEM), Version 2.0. http://www.omg.org/spec/SPEM/2.0/. (2008). (Last accessed: December 03, 2017)

[30] Object Management Group (OMG). 2017. Structured Assurance Case Metamodel (SACM), Version 2.0 – http://www.omg.org/spec/SACM/2.0/Beta1/. (2017)

[31] A. Ruiz, “A Harmonized Compositional Assurance Approach for Safety-Critical Systems”, Ph.D. Thesis, 2015

[32] B. Gallina, S. Kashiyarandi, H. Martin, R. Bramberger. (2014, July). Modeling a safety-and automotive-oriented process line to enable reuse and flexible process derivation. In Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International (pp. 504-509). IEEE

[33] B. Gallina, Towards Enabling Reuse in the Context of Safety-critical Product Lines. 5th International Workshop on Product LinE Approaches in Software Engineering (PLEASE), joint event of ICSE, Florence, Italy, May 19th, 2015

[34] Barbara Gallina, A Model-driven Safety Certification Method for Process Compliance, 2nd International Workshop on Assurance Cases for Software-intensive Systems (ASSURE), joint event of ISSRE 2014

[35] Øystein Haugen and Ommund Øgård. 2014. BVR - Better Variability Results. In System Analysis and Modeling: Models and Reusability - 8th International Conference, SAM 2014, Valencia, Spain, September 29-30, 2014. Proceedings. 1–15. https://doi.org/10.1007/978-3-319-11743-0_1

[36] Anatoly Vasilevskiy, Øystein Haugen, Franck Chauvel, Martin Fagereng Johansen, and Daisuke Shimbara. July 20-24, 2015, Nashville, TN, USA. The BVR tool bundle to support product line engineering. In Proceedings of the 19th International Conference on Software Product Line (SPLC ’15). https://doi.org/10.1145/2791060.2791094

[37] Vasilevskiy, A., & Haugen, Ø. (2014, September). Resolution of interfering product fragments in software product line engineering. In International Conference on Model Driven Engineering Languages and Systems (pp. 467-483). Springer, Cham.

[38] Øystein Haugen, Birger Møller-Pedersen, Jon Oldevik, Gøran K. Olsen, and Andreas Svendsen. [n. d.]. Adding Standardized Variability to Domain Specific Languages. In Proceedings of the 12th International Conference on Software Product Lines (SPLC ’08), Limerick, Ireland, September 8-12, 2008. https://doi.org/10.1109/SPLC.2008.25

[39] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report. Carnegie-Mellon University Software Engineering Institute

[40] R. Hawkins and T. Kelly, “A software Safety Argument Pattern Catalogue” University of York, Department of Computer Science Report YCS-2013-482 (2013)

[41] E. Denney and G. Pai, “A Lightweight Methodology for Safety Case Assembly”, in Computer Safety, Reliability and Security, vol. 7612, F. Ortmeier and P. Daniel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1-12

[42] Epsilon Transformation Language, https://www.eclipse.org/epsilon/doc/etl/. (Last accessed: January 15, 2018)

[43] Eclipse Process Framework (EPF) Composer, Installation, Introduction, Tutorial and Manual. https://www.eclipse.org/epf/general/EPF_Installation_Tutorial_User_Manual.pdf. (Last accessed: January 15, 2018)

[44] S. Mazzini, J. Favaro, S. Puri, L. Baracchi, “CHESS: an open source methodology and toolset for the development of critical systems”, Open Source Software for Model-Driven Engineering (OSS4MDE'16), October 3, 2016

[45] Šljivo, Irfan. "Facilitating Reuse of Safety Case Artefacts Using Safety Contracts." Licentiate Thesis. Mälardalen University. (2015).

[46] B. Gallina, L. Fabre. Benefits of Security-informed Safety-oriented Process Line Engineering. IEEE 34th Digital Avionics Systems Conference (DASC-34), Prague, Czech Republic, September 13-17, ISBN 978-1-4799-8939-3, 2015.

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

[48] VDA QMC Working Group 13 / Automotive SIG, Automotive SPICE, Process Reference and Assessment Model, Version 3.1, 2017, http://www.automotivespice.com

[49] B. McIsaac, "IBM Rational Method Composer: Standards Mapping." 2015.

[50] F. UL Muram, B. Gallina, and L. Gomez Rodriguez. 2018. Preventing Omission of Key Evidence Fallacy in Process-based Argumentations. In 11th International Conference on the Quality of Information and Communications Technology (QUATIC), Coimbra, Portugal, September 4-7, 2018.

[51] Regorous Process Designer is available under an evaluation license https://research.csiro.au/data61/regorous/

[52] J. P. Castellanos Ardila, B. Gallina, and F. Ul Muram, "Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models," in *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2018.

[53] G. Governatori, "Representing business contracts in RuleML," *Int. J. Coop. Inf. Syst.*, vol. 14, no. 02n03, pp. 181–216, 2005.

[54] J. P. Castellanos Ardila and B. Gallina, "Formal Contract Logic Based Patterns for Facilitating Compliance Checking against ISO 26262," in *1st Workshop on Technologies for Regulatory Compliance (TeReCom)*, 2017, pp. 65–72.

[55] Overall A&T Methodology, MBAT Deliverable D_WP2.1_2_2, available on www.mbat-artemis.eu

[56] Natasha Sharygina, Doron Peled; "A Combined Testing and Verification Approach for Software Reliability"; in Proc. of FME '01 (International Symposium of Formal Methods for Increasing Software Productivity), pp.611-628 Springer Verlag London, UK ©2001; ISBN:3-540-41791-5

[57] Pattern-Oriented Software Architecture Volume 1: A System of Patterns, Wiley, 1996

[58] The Elastic Stack, https://www.elastic.co/, 11/2018

[59] The Elasticsearch search engine, https://www.elastic.co/products/elasticsearch, 11/2018

[60] The Kibana data visualization tool, https://www.elastic.co/products/kibana, 11/2018

[61] Safety Architect, https://www.all4tec.com/safety-architect

[62] IEC 61508, International Electrotechnical Commission: Functional safety of electrical/electronic/ programmable electronic safety-related systems. (2010)

[63] WEFACT user manual and Installation Instructions, 2018, Available in the AMASS project repository: WEFACT_UserManual.docx and WEFACT_Installation_Instructions.docx

[64] G. Governatori, "The regorous approach to process compliance," in *IEEE 19th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW)*, 2015, pp. 33–40

[65] G. Governatori and S. Sadiq, "The Journey to Business Process Compliance," *Public Law*, pp. 1–32, 2009.

[66] S. Sadiq and G. Governatori, "Managing regulatory compliance in business processes," *Handb. Bus. Process Manag. 2 Strateg. Alignment, Governance, People Cult.*, pp. 265–288, 2015.

[67] M. Panunzio and T. Vardanega, "A component-based process with separation of concerns for the development of embedded real-time software systems", Journal of Systems and Software, Volume. 96, pp. 105–121, 2014.

Appendix A: Changes with respect to D6.7 (*)

New Sections:

Section	*Title*
2.1.6	Reuse and component-based system engineering
3.1.5	Compliance Checking Workflow (*)
3.3.2.1	Cross-domain variability management: an automotive/avionics product line (*)
3.4.1.1	Modelling of standards requirements and safety processes (*)
3.4.1.2	Detecting fallacies in process models (*)
3.4.1.3	Generating process-based argumentation (*)
3.5	Transformation of Standard's Requirement from EPF-C to Baseline Model OpenCert (*)
3.6	Compliance Mapping (*)
3.7	Compliance Checking (*)
3.10	Methodology to reuse Safety and Security analysis artefacts (*)
3.11	Model based testing methodologies (*)
3.12	Summary (*)
Appendix A	Changes with respect to D6.7 (*)

Sections whose number has changed:

Former Section No.	*New Section No.*	*Title*
3.2	3.8	Reuse Discovery
3.3	3.2	Reuse Assistance
3.4	3.3	Management of Families/Lines
3.5	3.4	Automatic Generation of Arguments
3.6	3.9	Representation of Safety Standards with Semantic Technologies

Modified Sections:

Chapter/Section	*Title*	*Change*
2.2.2	Tool Support Overview (*)	Safety Architect and Elasticsearch & Kibana added
3.2	Reuse Assistance (*)	Extension
3.3.1.1	Cross-concern variability management: an automotive SiSoPLE with EPF-C and BVR (*)	Minor updates
3.3.2	Product-related reuse via management of product lines (*)	Extension
3.3.3	Assurance case-related reuse via management of assurance case lines (*)	Extension
3.4.1	Automatic generation of process-based argumentation representing plans (*)	Detection of fallacies in process models
3.9	Representation of Safety Standards with Semantic Technologies (*)	New details added