

**ECSEL Research and Innovation actions (RIA)**



**AMASS**

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

**Design of the AMASS tools and methods for  
cross/intra-domain reuse (b)  
D6.3**

<b>Work Package:</b>	WP6: Cross/Intra-Domain Reuse
<b>Dissemination level:</b>	PU = Public
<b>Status:</b>	Final
<b>Date:</b>	30 July 2018
<b>Responsible partner:</b>	Barbara Gallina (MAELARDALENS HOEGSKOLA)
<b>Contact information:</b>	<a href="mailto:barbara.gallina@mdh.se">barbara.gallina@mdh.se</a>
<b>Document reference:</b>	AMASS_D6.3_WP6_MDH_V1.0

PROPRIETARY RIGHTS STATEMENT

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

## Contributors<sup>1</sup>

Names	Organisation
Barbara Gallina, Julieth Patricia Castellanos Ardila, Muhammad Atif Javed, Irfan Sljivo, Faiz Ul Muram, Shankar Iyer	MAELARDALENS HOEGSKOLA (MDH)
Anna Carlsson	OHB Sweden (OHB)
Frank Bastuebner and Andreas Preussger	INFINEON (IFX)
Norbert Bartsch and Vladislav Gribov	Lange (LAN)
Jose Luis de la Vara, Jose María Álvarez, Pablo Sánchez, Elena Gallego, Valentín Moreno, Manuela Alejandres, Fabio di Ninno, Miguel Angel Rozalen	Universidad Carlos III de Madrid (UC3)
Borja López, Luis Alonso	The REUSE Company (TRC)
Helmut Martin, Robert Bramberger	Virtual Vehicle (ViF)
Detlef Sholle, Staffan Skogby, Samer Medawar	Alten Sweden (ALT)
Michael Soden, Jan Mauersberger	ANSYS medini Technologies (KMT)
Gabriel Pedroza, Morayo Adedjouma	Commissariat a l'énergie Atomique et aux Energies Alternatives (CEA)
Huascar Espinoza, Alejandra Ruiz, Angel López	Tecnalia Research & Innovation (TEC)
Stefano Puri	INTECS (INT)
Thomas Gruber	Austrian Institute of Technology (AIT)
Marc Sango	ALL4TEC (A4T)

## Reviewers<sup>2</sup>

Names	Organisation
Silvia Mazzini (Peer reviewer-D6.2)	INTECS (INT)
Erwin Schoitsch (Peer reviewer-D6.2, D6.3)	Austrian Institute of Technology (AIT)
Cristina Martinez (Quality Manager-D6.2, D6.3)	Tecnalia Research & Innovation (TEC)
Jose Luis de la Vara (TC review-D6.2, D6.3)	Universidad Carlos III de Madrid (UC3)
Andrea Musone (Peer reviewer D6.3)	INTECS (INT)
Garazi Juez Uriagereka (TC review)	Tecnalia Research & Innovation (TEC)

<sup>1</sup> The list includes the contributors to of D6.2, which is evolved in D6.3

<sup>2</sup> The list includes the reviewers of D6.2, which is evolved in D6.3



# TABLE OF CONTENTS

<b>Executive Summary .....</b>	<b>10</b>
<b>1. Introduction (*) .....</b>	<b>11</b>
<b>2. Recap concerning industrial needs with respect to STO4 (*) .....</b>	<b>13</b>
2.1 Industrial needs with respect to process engineering.....	13
2.2 Industrial needs with respect to product engineering .....	14
2.3 Industrial needs with respect to assurance case engineering .....	15
<b>3. AMASS vision for cross/intra domain reuse.....</b>	<b>17</b>
<b>4. Conceptual solution.....</b>	<b>19</b>
4.1 Process-related reuse .....	19
4.1.1 Process-related macro and micro (reusable) elements (*) .....	19
4.1.2 Summary of previously conceived and validated conceptual solutions .....	21
4.2 Product-related reuse.....	21
4.2.1 Recap on challenges related to product reuse .....	21
4.2.2 Product-related macro and micro (reusable) elements .....	28
4.2.3 Summary of previously conceived and validated conceptual solutions .....	29
4.3 Assurance case-related reuse .....	29
4.3.1 Assurance case-related macro and micro (reusable) elements .....	30
4.3.2 Summary of previously conceived and validated conceptual solutions .....	30
<b>5. Design Level Solution.....</b>	<b>32</b>
5.1 Reuse discovery.....	32
5.1.1 Methodology to represent system artefacts.....	33
5.1.2 Architecture and Operations to support reuse discovery .....	35
5.1.3 Design of a research method to evaluate a reuse discovery process .....	37
5.1.4 The reuse discovery process in AMASS.....	38
5.1.5 Definition of an interface for reuse discovery (*) .....	38
5.2 Reuse assistance (*) .....	45
5.2.1 Cross-system reuse scenario .....	48
5.2.2 Cross-standard reuse scenario.....	49
5.3 Management of families/lines .....	52
5.3.1 Base Variability Resolution .....	52
5.3.2 Process-related reuse via management of process lines .....	54
5.3.3 Product-related reuse via management of product lines (*) .....	66
5.3.4 Assurance case-related reuse via management of case lines (*) .....	81
5.3.5 Anti-Sisyphus: (3+1)-D Reuse and Impact Analysis via UMA, CHESSML, CACM, and BVR (*).....	85
5.4 Product-related reuse via MDE and meta-modelling: focus on analysis artefacts .....	88
5.5 Product-related reuse: focus on safety and security analysis artefacts (*).....	89
5.6 Conceptual approach on product reuse (*) .....	92
5.6.1 Reuse using a Model-based Integrated Safety Analysis Approach .....	93
5.6.2 Implementation Approach and Use Cases .....	94
5.7 Model Based Testing for exploring the benefits of re-use of development cycles (*).....	95
5.7.1 Automated Model Based Testing (*) .....	95
5.8 Approach on impact analysis and delta analysis based on data indices using Elasticsearch (*) .....	96
5.8.1 Achievements in P1.....	97
5.8.2 Challenges.....	98
5.8.3 Way forward / next step .....	99
5.9 Automatic generation of process-based arguments .....	101



5.9.1	Generating Process-based Argumentation Representing Plans (*)	101
5.9.2	Generating Process-based Argumentation Representing Executed Processes (*)	108
5.10	Automatic generation of product-based arguments	111
5.10.1	Argument-fragment generation at the architectural pattern level (*)	111
<b>6.</b>	<b>Implementation Solution for Cross/intra domain reuse: a way forward</b>	<b>113</b>
<b>7.</b>	<b>AMASS vision for compliance management (*)</b>	<b>115</b>
<b>8.</b>	<b>Conceptual solution for compliance checking</b>	<b>117</b>
8.1	Essential Background information	117
8.1.1	Defeasible Logic	117
8.1.2	Formal Contract Logic	118
8.1.3	Regorous Process Designer	120
8.1.4	Property Specification Patterns for Finite-State Verification (*)	123
8.1.5	EPF Composer metamodels (*)	124
8.1.6	Regorous metamodels (*)	125
8.2	Pioneering compliance checking in AMASS	125
8.2.1	Exploring the usage of defeasible logic and compliance by design.	125
8.2.2	Exploring the usage of REGOROUS for compliance checking	125
8.2.3	Exploration conclusions	134
8.3	Conceptual solution	135
<b>9.</b>	<b>Potential design solutions for the compliance checking</b>	<b>136</b>
9.1	Proposals for adapting Regorous to the needs of AMASS	136
9.2	Creating an AMASS process compliance checker from scratch	138
9.3	Pros and cons of the architectural design solutions	138
<b>10.</b>	<b>AMASS design solution for compliance checking (*)</b>	<b>140</b>
10.1	Safety compliance patterns	140
10.1.1	Safety compliance patterns	140
10.1.2	ISO 26262-related compliance patterns identification	140
10.1.3	ISO 26262-related compliance patterns definition	141
10.1.4	ISO 26262-related compliance patterns instantiation	142
10.2	Modelling SPEM 2.0-compatible process models for compliance checking	143
10.2.1	Mechanisms to annotate software process models	144
10.2.2	Modelling and annotating a small example from ISO 26262	144
10.3	Generating Regorous inputs	147
10.3.1	Generating the rule set	147
10.3.2	Generating the structural representation of the process	148
10.3.3	Generating the Compliance Effect Annotations	150
10.3.4	Model checkable for compliance: an example for ISO 26262	151
<b>11.</b>	<b>Implementation solution for compliance checking: a way forward</b>	<b>154</b>
<b>12.</b>	<b>Conceptual solution for ontology-based mapping (*)</b>	<b>155</b>
12.1	Representation of Safety Standards with Semantic Technologies Used in Industrial Environments (*)	155
12.2	Semantic Analysis of Safety Standards (*)	159
<b>13.</b>	<b>Implementation solution for the ontology-based compliance management vision: a way forward</b>	<b>160</b>
<b>14.</b>	<b>Metrics for reuse</b>	<b>161</b>
14.1	GQMPS for process-related reuse	161
14.1.1	GQMPS	161
14.1.2	GQM + Strategies Model for the evaluation of families of processes (*)	161
14.2	GQMPS for product-&-assurance case related reuse (*)	163





<b>15. Conclusion .....</b>	<b>164</b>
<b>Abbreviations and Definitions .....</b>	<b>165</b>
<b>References.....</b>	<b>168</b>
<b>Appendix A .....</b>	<b>175</b>
<b>Appendix B .....</b>	<b>176</b>
<b>Appendix C. Changes with respect to D6.2 (*) .....</b>	<b>184</b>



## List of Figures

Figure 1.	AMASS Vision for Cross/Intra Domain Reuse .....	18
Figure 2.	UMA-based solution partly supporting process reuse.....	21
Figure 3.	From out of context to in-context: the challenge of component reuse .....	23
Figure 4.	From out of context to in-context: focus on the interfaces .....	24
Figure 5.	FMEDA hierarchy within the automotive domain .....	25
Figure 6.	Conceptual overview of common components from a vendor's perspective .....	25
Figure 7.	CHESSML-based solution partly supporting component reuse .....	29
Figure 8.	SACM-based assurance case metamodel.....	31
Figure 9.	Layers of an ontology-driven approach to implement a Knowledge-Centric Systems Engineering strategy.....	33
Figure 10.	A system Knowledge Repository structure. ....	36
Figure 11.	UML Class Diagram of the OSLC Knowledge Management Resource Shape. ....	40
Figure 12.	Decision tree to expose existing operations in a REST-oriented fashion.....	43
Figure 13.	Building blocks of the functional architecture and technology for an OSLC KM environment. ....	45
Figure 14.	Reuse Assistant: Scope of Reuse in Assurance and Certification .....	46
Figure 15.	Reuse Assistant: Proposed Reuse Approaches .....	47
Figure 16.	Reuse Assistant: Architecture of new functionalities (in green) in OpenCert tooling .....	48
Figure 17.	Reuse Assistant: Components decomposition of Reuse Assistant .....	48
Figure 18.	Reuse Assistant mock-up: Cross-system reuse scenario.....	49
Figure 19.	Reuse Assistant mock-up: Supporting equivalence mapping process.....	50
Figure 20.	Reuse Assistant: Showing reuse opportunities based on equivalence relations .....	51
Figure 21.	Fragment substitutions exemplification, taken from [40] .....	52
Figure 22.	Editors' architecture.....	53
Figure 23.	Third party integration .....	54
Figure 24.	Architecture of the seamless integrator plugin enabling process-related variability management ..	55
Figure 25.	Models interplay enabling management of process lines .....	56
Figure 26.	UMA-based model of the ECSS process fragment.....	57
Figure 27.	VSpec model regarding a portion of ECSS-E-ST-40C .....	57
Figure 28.	Resolution model.....	58
Figure 29.	Realization model .....	58
Figure 30.	Backward propagation of the changes onto the original model .....	59
Figure 31.	Recommendation table from ISO 26262-4.....	61
Figure 32.	Variability management, ASIL B and Recommendation level “+” are determined, SecL is not defined .....	62
Figure 33.	VSpec diagram: Concept phase of an integrated safety and security process (ASIL:=B) .....	63
Figure 34.	Resolution diagram: Concept phase of an integrated safety and security process (ASIL:=B) .....	64
Figure 35.	Realization diagram and imported EPF-C model (Placement for Fragment Substitution in red) .....	64
Figure 36.	EPF-C model before (left) and after replacement of FTA (right) .....	65
Figure 37.	Process for verification in WEFACT .....	65
Figure 38.	Models interplay enabling management of product lines .....	66
Figure 39.	Architecture of the seamless integrator plugin enabling product-related variability management..	67
Figure 40.	ACS component model given in CHESSML .....	68
Figure 41.	VSpec Model regarding Attitude Control System (ACS).....	69
Figure 42.	Resolution Model regarding Attitude Control System (ACS).....	69
Figure 43.	Realization Model regarding Attitude Control System (ACS) .....	70
Figure 44.	CHESS Views .....	71
Figure 45.	SSTH (Sun SensorsTHrusters) .....	72
Figure 46.	SSRW (Sun Sensors Reaction Wheels) .....	72



Figure 47.	STTH (Star Tracker Thrusters) .....	73
Figure 48.	STRW (Star Tracker Reaction Wheels) .....	73
Figure 49.	SSTH, SSRW, STTH and STRW Configurations.....	74
Figure 50.	Intra-Domain Variability Modelling .....	75
Figure 51.	Intra-Domain Variability Resolution .....	76
Figure 52.	Intra-Domain Variability Realization .....	76
Figure 53.	Infineon Automotive Electronic Computing Unit Model .....	79
Figure 54.	Lange Aviation Central Computing Unit Model.....	80
Figure 55.	VSpec representing the cross-domain product line.....	81
Figure 56.	Variability Resolution for Automotive .....	81
Figure 57.	Automotive ECU realization .....	81
Figure 58.	Models interplay enabling management of assurance case lines .....	82
Figure 59.	Argumentation for FLEDS in OpenCert .....	83
Figure 60.	VSpec model regarding FLEDS .....	84
Figure 61.	Resolution models regarding FLEDS .....	84
Figure 62.	Realization model regarding FLEDS .....	85
Figure 63.	VSpec Model regarding ECSS-related SW Development Process and Attitude Control System (ACS) .....	86
Figure 64.	Resolution Model regarding ECSS-related SW Development Process and Attitude Control System (ACS).....	86
Figure 65.	Models interplay enabling management of process product and assurance case lines.....	87
Figure 66.	Impact analysis and change propagation in families/lines .....	88
Figure 67.	MDE-based seamless safety-security analysis targeting reusable products design .....	89
Figure 68.	EMF Diff/Merge Principle.....	90
Figure 69.	Diff/Merge implementation in Safety Architect.....	90
Figure 70.	Merge of two Safety Architect models .....	91
Figure 71.	Differences between two models.....	91
Figure 72.	Import from CHES to Safety Architect.....	92
Figure 73.	Overview of the Farkle implementation of model-based testing.....	96
Figure 74.	Elasticsearch overview .....	97
Figure 75.	Interaction between P1 indexing tool, search app and Kibana dashboard .....	98
Figure 76.	Relationships denormalization .....	99
Figure 77.	Score information as meta-data .....	100
Figure 78.	Utilization of Elasticsearch based APIs.....	101
Figure 79.	Process-based arguments generation (Planning Phase).....	102
Figure 80.	Overview of the proposed method.....	102
Figure 81.	Requirements and Process modelling in EPF Composer.....	104
Figure 82.	Result after detecting omission of key evidence fallacies.....	106
Figure 83.	Generated argumentation model and diagram .....	108
Figure 84.	Process-based arguments generation (Execution Phase) .....	109
Figure 85.	Product-based argument generation.....	111
Figure 86.	Compliance Management Vision, adapted from [155]. .....	116
Figure 87.	Abstract framework of Regorous, taken from [72].....	121
Figure 88.	Regorous architecture.....	122
Figure 89.	Main components of SPINdle reasoner. Taken from [80].....	123
Figure 90.	Property Specification Patterns Hierarchy .....	124
Figure 91.	Product development at the software level.....	126
Figure 92.	Example of requirement for ISO 26262-software unit design and implementation .....	127
Figure 93.	Trace 1 of the software unit design and implementation phase.....	128
Figure 94.	A weakly compliant process checked by Regorous .....	129
Figure 95.	Modelling of conditional requirements for ISO 26262: Software unit design and specification .....	130
Figure 96.	Terms required for modelling rules for ISO 26262-Software unit design and Specification.....	131



Figure 97.	Rules specification for ISO 26262-Software unit design and specification .....	132
Figure 98.	Definition of the traces for the process Software unit design and specification .....	133
Figure 99.	Complete model of ISO 26262-Software unit design and specification .....	134
Figure 100.	Adding EPF Composer + rule editor .....	136
Figure 101.	Replacing Activiti BPMN 2.0 with EPF Composer + rule editor .....	137
Figure 102.	Replacing BPMN with EPF Composer + rule editor + extended ComputeObligations/CheckCompliance components .....	137
Figure 103.	AMASS Process Compliance Designer.....	138
Figure 104.	Requirement that represents the initiation of the software unit design and implementation (sub- )phase.....	142
Figure 105.	Requirement that represents the selection of disjoint implementation strategies .....	142
Figure 106.	Requirement that represents the selection of mandatory notations .....	143
Figure 107.	AMASS Compliance Checking Vision.....	143
Figure 108.	Rules required for compliance checking of a small example in ISO 26262 .....	145
Figure 109.	Standard's requirements plugin .....	145
Figure 110.	Specification of rule 3.1.....	145
Figure 111.	Rule set specification .....	146
Figure 112.	Process elements plugin.....	146
Figure 113.	Annotated task .....	146
Figure 114.	Activity Diagram of the Software Unit Design Process .....	147
Figure 115.	Algorithm for Obtaining the Rule Set.....	148
Figure 116.	Algorithm for Obtaining the Process Structure .....	150
Figure 117.	Algorithm for Obtaining the Compliance Effects Annotations .....	151
Figure 118.	Rule set generated.....	152
Figure 119.	Process structure generated .....	153
Figure 120.	Compliance annotations generated.....	153
Figure 121.	Excerpt of the metamodel for the specification of safety compliance needs.....	156
Figure 122.	Ontology layers in KM.....	157
Figure 123.	Example of specification of a standard's information with Knowledge Manager.....	158
Figure 124.	SoPLE-targeted GQM+ Strategies Model .....	162
Figure 125.	Expanded version of Figure 40 .....	177
Figure 126.	Expanded version of Figure 41 .....	178
Figure 127.	Expanded version of Figure 45 .....	179
Figure 128.	Expanded version of Figure 46 .....	180
Figure 129.	Expanded version of Figure 47 .....	181
Figure 130.	Expanded version of Figure 48 .....	182
Figure 131.	Re-configured argumentation fragment.....	183



## List of Tables

Table 1.	Summary of STO4-related Use Cases [D1.1], where y stands for low priority, Y stands for high priority and N/A stands for Not Applicable (in the context of AMASS) .....	13
Table 2.	Summary of STO4-related high-level requirements and related CSs .....	16
Table 3.	Common operations definition for the System Knowledge Base .....	36
Table 4.	Operations for the management of the System Knowledge Base and the System Assets Store .....	37
Table 5.	OSLC Resource Shapes for OSLC Defined Resources within the KM Domain .....	40
Table 6.	Mapping of WSDL/SOAP operations to OSLC concepts. ....	43
Table 7.	Delegated operations for an OSLC KM provider (SKB). ....	44
Table 8.	Delegated operations for an OSLC KM provider (SAS). ....	44
Table 9.	Mapping between avionics and automotive regulations. ....	60
Table 10.	Mapping between process model, argumentation model and GSN .....	107
Table 11.	Mapping between WEFACT and OpenCert model elements .....	110
Table 12.	STO4-Reuse Assistant as well as Product/Process/Assurance Case Line Specification + semi-automatic generation of arguments .....	113
Table 13.	STO4-Product/Process/Assurance Case Line Specification + semi-automatic generation of arguments .....	114
Table 14.	Property Specification Patterns Scope .....	123
Table 15.	Pros and cons of the architectural design solutions .....	139
Table 16.	Mapping of the patterns scope into FCL rule notation .....	141
Table 17.	Annotated process description. ....	147
Table 18.	Mapping Elements from UMA to the Rule Set .....	148
Table 19.	Mapping Elements from UML Diagram to a BPMN and Canonical Process. ....	148
Table 20.	Mapping Elements from UMA/UML Metamodel to the Compliance Check. ....	150
Table 21.	Compliance checking .....	154
Table 22.	Ontology-based compliance management .....	160



## Executive Summary

This deliverable (D6.3 - Design of the AMASS tools and methods for cross/intra-domain reuse (b)) is conceived as an update<sup>3</sup> of D6.2 - Design of the AMASS tools and methods for cross/intra-domain reuse (a) [18], which was delivered as a confidential document. D6.3 is the final and public outcome of Task 6.2 Conceptual Approach for Cross-Domain and Intra-Domain Reuse.

This deliverable presents the final design of the AMASS cross/intra-domain reuse vision, embracing three dimensions: process, product, and assurance case. It also targets the final design of the strengthened vision for compliance management.

The final design of both visions (cross/intra domain and compliance management) embraces functionalities related to reuse assistance, semantics-based mapping of standards, specification of families of processes/products/assurance cases, and logic-based automatic compliance checking. The final design is conceived as an extension of the AMASS Reference Tool Architecture, initially specified in D2.2 [4], then in D2.3 [5], and finally in D2.4 [6]. The design of both visions consists of the final specification of the architectural solution and of the identification of the extension of the AMASS Common Assurance and Certification Meta-model (CACM), to support the Scientific Technical Objective (STO) regarding Cross/Intra-Domain Reuse, STO4, and compliance management.

Relations with D3.3 [10], D4.3 [13], and D5.3 [16] are explained, whenever reuse-related concerns involve other work packages.

The solutions, presented in this deliverable, will guide the implementation of the third iteration of the AMASS prototype, P2, in Task 6.3 (Implementation for cross/intra-domain reuse) for what regards the cross/intra-domain reuse features of the AMASS platform as well as its strengthened compliance management features.

Finally, Task 6.4 (Methodological Guidance for cross/intra-domain reuse) will build upon the results identified in this deliverable to provide methodological guidance to the AMASS end-users for the application of the cross/intra-domain reuse solution.

---

<sup>3</sup> The sections modified with respect to D6.2 have been marked with (\*), then the details about the differences and modifications are provided in Appendix C: Document changes with respect to D6.2 (\*)



## 1. Introduction (\*)

In the context of Cyber Physical Systems (CPSs), the pace of assurance and certification will be determined by the ability of both industry and certification/assessment authorities to overcome technical, regulatory, and operational challenges. A key regulatory-related challenge is faced when trying to reuse CPS products qualified or certified for one application domain in another one. This challenge emerges because different domains are constrained by different standards and the full assurance and certification process must be applied as if it were a totally new product, thus reducing the return on investment of the reuse decisions. Similarly, reuse is often hindered even within the same domain, when trying to reuse CPS products proven in use in one project in another, where assumptions change (e.g., about the environment), and sometimes also the criticality level. The increased connectivity of CPSs also contributes to hindering their reuse. Security-uninformed CPS products, developed for non-connected safety-critical systems, require new solutions for enabling cross-concern reuse when concern-specific regulations are in place and concern-specific threats impact CPSs' dependability.

WP6 aims at addressing these challenges related to cross and intra-domain reuse as well as cross-concern reuse. More specifically, this deliverable (D6.3) documents the work conducted in the scope of Task 6.2, which mainly addresses the design of the AMASS tools and methods related to: Goal 2 (G2), the corresponding project objective O3, and the project Scientific and Technical Objective STO4. G2, O3 and STO4 are recalled here to make the deliverable self-contained.

**G2** demonstrates a potential reuse of assurance results (formerly either qualified or certified), leading to 40% of cost reductions for component/product (re)certification/qualification activities.

**O3** consolidates a cross-domain and intra-domain assurance reuse approach to improve the mutual recognition agreement of compliance approvals and to help assessing the return of investment of reuse decisions.

**STO4** focuses on Cross/Intra-Domain Reuse and is constituted of three sub-objectives:

- **Semantics-based Standards Equivalence.** This sub-objective is expected to solve or at least reduce the terminological and semantic inconsistencies, which are present across different application domains and which hinder an efficient reuse of assurance artefacts<sup>4</sup>.
- **Mapping, Reuse Assistant (Cross/Intra Domain).** This sub-objective is expected to ease the understanding of the role played by each activity and artefact in the overall assurance effort.
- **Product/Process/Assurance-Case Line Specification.** This sub-objective is expected to ease variability management within interconnected families of products (product lines), processes (process lines), and Assurance Cases (assurance case lines).

WP6-Task 6.2 contributes to the achievement of these sub-objectives as follows:

- **Regarding semantics-based Standards Equivalence Mapping,** AMASS extends the OPENCOS functionality for mapping between standards by supporting ontology-based analysis for the creation of the maps.
- **Regarding the Reuse Assistant (Cross/Intra Domain),** AMASS supports users in evaluating whether reuse of the assurance assets is feasible (appropriate) or determining what further analysis is required to justify claims of compliance. The Reuse Assistant will benefit from the compositional argument approach, which was developed by SafeCer and OPENCOS to achieve a characterization of pre-existing argument modules in order to meet the intent of the applicable standards.

---

<sup>4</sup> For sake of clarity, it is worth noting that in this document *artifact* and *artefact* co-exist. The AMASS documents are written in UK English (*artefact*). However, OMG specifications make use of US English and the OMG SACM specification contains the meta-class *artifact*.



- **Regarding Product/Process/Assurance Case Line Specification**, AMASS develops a systematic approach for dealing with software/hardware variability management, but also with process and assurance case-related variability. The AMASS project focuses on extending and integrating the current methods (developed within SafeCer) in order to manage, for instance, the ripple effects (i.e., the impact) on processes as well as assurance cases, as results of changes in product requirements.

In addition, WP6-Task 6.2 is responsible for further developing the Compliance management building block, which was delivered as part of the AMASS first prototype, called Core. In particular, the expected development within T6.2 consists of elaborating solutions for enabling automatic process-based argumentation generation, ontology-based compliance management, and compliance checking.

Based on the proposed solutions, a way forward, enabling the implementation of the AMASS visions regarding compliance management and reuse, is given.

The rest of the deliverable is organized as follows:

- Chapter 2 gives a recap concerning industrial needs with respect to STO4.
- Chapter 3 gives the AMASS vision regarding cross/intra domain reuse.
- Chapter 4÷6 present the conceptual solution, the design solution, and the way forward for the implementation solution regarding the AMASS vision for cross/intra domain reuse.
- Chapter 7 provides the AMASS extended vision regarding compliance management.
- Chapters 8÷11 develop such vision by presenting solutions at conceptual, design, and implementation levels, focusing on semi-automatic compliance checking.
- Chapters 12÷13 further develop such vision by presenting solutions at conceptual, design, and implementation levels, focusing on ontology-based solutions.
- Chapter 14 proposes a set of metrics aimed at measuring the advantage, which could be gained through adoption/application of the proposed solution for cross/intra domain reuse.
- Finally, Chapter 15 draws some conclusions.



## 2. Recap concerning industrial needs with respect to STO4 (\*)

AMASS expected results will be benchmarked by eleven Case Studies:

- CS1: (Industrial Automation) Industrial and Automation Control Systems;
- CS2: (Automotive) Advanced driver assistance function with electric vehicle sub-system;
- CS3: (Automotive) Collaborative automated fleet of vehicles;
- CS4: (Space) Design and safety assessment of on-board software applications;
- CS5: (Railways) Platform Screen Doors Controller;
- CS6: (Railways) Automatic Train Control & Interlocking Formal Verification;
- CS7: (Avionics & Automotive) Safety assessment of multi-modal interactions in cockpits;
- CS8: (Automotive) Telematics function;
- CS9: (Air Traffic Management) Safety-Critical SW Life-cycle of a Monitoring System for Navigational Aid (NavAid);
- CS10: (Space) Certification basis to boost the usage of Multiprocessor System-on-Chip (MPSoC) architectures;
- CS11: (Space) Design and efficiency assessment of model-based Attitude and Orbit Control software development.

All case studies except CS7 and CS2 are single-domain centred. Thus, their main interest is intra domain reuse. CS7 focuses on avionics, however it presents scenarios related not only to intra domain reuse but also scenarios related to the exploration of cross-domain reuse with focus on reuse of process-related information from automotive to avionics regarding hardware COTS. Similarly, CS2 focuses on automotive, however it presents also one scenario related to the exploration of cross-domain reuse with focus on reuse of product-related information from avionics to automotive. This scenario however is not expected to be developed in detail. It is only expected to be a very preliminary learning experience.

As it was initially elicited in D1.1 [1] and then refined (due to changes within the consortium), the above-listed eleven case studies focus on the different dimensions of reuse. Table 1 summarizes the industrial needs with respect to STO4.

**Table 1.** Summary of STO4-related Use Cases [D1.1], where y stands for low priority, Y stands for high priority and N/A stands for Not Applicable (in the context of AMASS)

STO4 Intra Domain Reuse	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	CS11
Product	Y	Y	Y	N/A	N/A	Y	Y	N/A	y	Y	Y
Process	y	y	Y	N/A	N/A	Y	y	y	y	N/A	Y
Assurance Case (Product)	Y	Y	Y	N/A	N/A	Y	Y	N/A	y	Y	Y
Assurance Case (Process)	y	y	Y	N/A	N/A	Y	N/A	y	y	N/A	Y

From Table 1, it emerges that nine CSs are related to STO4. Their specific needs with respect to the different dimensions are recalled in the following subsections.

### 2.1 Industrial needs with respect to process engineering

This subsection recalls the industrial needs with respect to process engineering. More specifically, the following bulleted list recalls the specific needs stemming from the following AMASS Case Studies (CSs), as taken from former comprehensive list:

- CS1: Given the interest in product reuse and in compliance management, indirectly an interest in reuse of process-related information emerges.
- CS3: Reuse or enhancement of current safety methods (HARA) for other concerns, such as cyber-



security (TARA).

- CS6: Reuse of compliance management artefacts, e.g. safety plans.
- CS8: Methodology for handling interplay between concerns and/or re-use between concerns for multi-concern assurance and assessment for multiple standards.
- CS11: Reuse of process-based engineering and assurance artefacts.

In D6.1 [17], the reuse scenarios related to the process-dimension were identified. They are briefly recalled here:

1. **Regulatory jurisdiction:** Critical systems may operate in places where different jurisdictions apply, for example a plane landing in different countries. In this case, different jurisdictions apply to the same product/component and **the certification artefacts generated for one jurisdiction can be used in order to achieve compliance with other jurisdiction(s)**. When components are expected to be used in different countries, the different jurisdictions of each country shall be taken into account during the component design.
2. **Communities of practice:** Reuse of the methodologies and practices related to one or more activities mentioned in a standard and shared between different communities with the same objectives.

Based on D1.1 [1], none of the CSs focuses on regulatory jurisdictions. The main focus is on communities of practice. Despite the absence of a specific CS focusing on regulatory jurisdictions, a brainstorming session was held, during a meeting, to identify cases where regulatory jurisdictions may play a decisive role. Such a session was held in order to challenge our design solutions and make them robust in case of additive or conflicting requirements stemming from different jurisdictions. Specifically discussed was a hypothetical case of a Bi-Standards ERTMS (European Rail Traffic Management System)/TVM (Transmission Voie-Machine, English: track-to-train transmission, which is a form of in-cab signalling) on-board system that crosses French/Swiss border. In this case, depending on the country, the on-board system is running two conflicting behaviours, prescribed for the same system and appropriate context switching has to be guaranteed.

The Swiss national requirement - OFT (Office Fédéral des Transports): CH-TSI LOC&PAS-022 [113] is in conflict with the French national requirement - EPSF (Établissement Public de Sécurité Ferroviaire): SAM S 706 [114]. To reduce time and cost while spotting inconsistencies, new means are needed. More specifically, specification means, connecting requirements stemming from standards and architectural requirements, are essential. For instance, the solution presented in Chapter 5.3.5 is expected to serve this purpose, where constraints can be specified to limit the inclusion/exclusion of functionalities depending on specific choices (e.g., contextual choices related to jurisdictions).

Essential might also be automatic compliance checking methods able to identify contradictions and support standardization bodies to solve the issues at the source. For instance, the solution presented in Chapter 10 is expected to serve this purpose.

## 2.2 Industrial needs with respect to product engineering

This subsection recalls the industrial needs with respect to product engineering. More specifically, the following bulleted list recalls the specific needs stemming from the following AMASS Case Studies (CSs), as taken from former comprehensive list:

- CS1: Reuse in the case of product upgrades and product families.
- CS2: Reuse of self-assessment artefacts when undergoing partial changes, be it a variant of a product family or a change of components due to new suppliers.
- CS6: Reuse of product evidence such as formal proofs.
- CS7: Reuse of the existing artefacts (Automated safety assessment results, formal verification results) within aerospace domain.
- CS8: Reuse of e.g. analysis and verification results between different concerns (e. g., safety and security) in a multi-concern assurance case.



- CS10 as well as CS11: Reuse of pre-qualified components, or components that have been certified in a previous space mission.

In D6.1 [17], the reuse scenarios related to the product were identified. In this subsection, they are briefly recalled:

1. **(Same project) Upgrade – new feature:** A component associated with a particular hardware and/or software will include new features that the previous component did not have. There is a basic component that will include a new feature in its next version.
2. **(Same project) Upgrade – Enhance performance:** A component associated with a particular hardware and/or software will be modified as part of its maintenance and will keep the same functionalities as the previous version but with enhanced performance.
3. **Similar project:** A component is reused and integrated into a new system with the same context and domain as previously used. The functionalities needed in both projects are the same and so the component can be straightforwardly reused.
4. **Similar Project - Product Lines:** A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and are developed from a common set of assets in a prescribed way.
5. **Different project - same domain:** A component developed for a specific project in a certain domain is reused in another project with a different context<sup>5</sup>, but in the same domain. The operational environments and/or systems in which the component is integrated might be different.
6. **Different project - different domain:** General-purpose components (e.g., operating systems) may be reused not only in projects from the same domain but also in projects from different domains.
7. **(C)OTS:** In this case, the reuse is of a component described as “(Commercial) Off-The-Shelf”. COTS components are usually general-purpose ones that can apply to different domains and purposes. OTS (without leading “Commercial” in the term) are instead developed in house and may or may not cross the original domain.

In the scientific literature, COTS<sup>6</sup> (if software) are identified as Software of Unknown Pedigree (SOUP) when it is not proven (documented) that their development has followed the best practices mandated by the applicable domain standards. Components with a “pedigree” are for instance ISO 26262-compliant SEooC (Safety Elements out of Context) [90].

## 2.3 Industrial needs with respect to assurance case engineering

This subsection recalls the industrial needs with respect to assurance case engineering. More specifically, the following bulleted list recalls the specific needs stemming from the following AMASS Case Studies (CSs), as taken from former comprehensive list:

- CS6: Reuse of assurance cases’ structure.
- CS7: Reuse of the existing artefacts (Safety assessment argumentation methods).
- CS9: Automatic generation of reports, checklists and evidences to support the certification. Automatic check to verify that all the objectives, stated in the standards, have been satisfied.
- CS11: Systematic reuse of product-based assurance artefacts.

In D6.1 [17], the reuse scenarios related to the assurance case were identified. In what follows, they are briefly recalled:

---

<sup>5</sup> Note that the different contexts in which a differently configured component is deployed could to some extent be addressed via product line techniques. The set of differently configured components may also be interpreted as a product line and thus product line best practices can be used.

<sup>6</sup> Note that a COTS could be addressed via product line best practices since a COTS could to some extent be seen as a set of components with different configuration parameters.

1. **Argumentation – Patterns:** Assurance case patterns are considered as one of the main approaches for managing reuse of assurance. An assurance case pattern provides a means of explicitly and clearly documenting the structure of common reasoning as found in assurance cases, and it promotes the reuse of best practices for assurance.
2. **Argumentation – Modules:** Assurance case modules are parts of an overall assurance case containing part of an argument and relevant citations of evidence.  
For instance, when contributing to an argument aimed at showing process compliance, an assurance case module may correspond to an interrelated set of assurance activities, scope of responsibilities of a particular engineering organisation, or well-defined sub-system or equipment used within the overall CPS platform.

Given the elicited needs, within D2.1 [3], high-level requirements were specified aimed at addressing such needs. Table 2 recalls such requirements. Note that in the following chapters, at the end of each designed solution, some tables detailing the covered requirements and their descriptions are presented.

**Table 2.** Summary of STO4-related high-level requirements and related CSs

	Requirement (as it was formulated in D2.1 [3])	Case Study (CS)
<b>STO4 Intra Domain Reuse</b>	Intra-Domain, Intra standard, Reuse Assistance	CS1, CS9
	Intra-Domain, Cross version, Reuse Assistance	CS1
	Reusable off the shelf components	CS1, CS11
	Intra-Domain, Intra standard, Different Stakeholders, Reuse/Integration Assistance	CS2, CS11, CS10
	The AMASS tools must support variability management at process level	CS3, CS7, CS11, CS8
	The AMASS tools must support variability management at product level	CS1, CS2, CS11
	The AMASS tools must support variability management at assurance case level	CS11
	Semi-automatic generation of product arguments	CS6, CS11, CS3
	Semi-automatic generation of process arguments	CS6, CS9
	Requirement (as it was formulated in D2.1 [3])	Case Study (CS)
<b>STO4 Cross Domain Reuse</b>	Cross-Domain Reuse Assistance	CS7, CS2
	The AMASS tools must support variability management at process level	CS7
	The AMASS tools must support variability management at product level	CS2
	Semantics-based mapping of standards	CS7



### 3. AMASS vision for cross/intra domain reuse

The vision of AMASS for cross/intra domain reuse is exemplified in Figure 1, which is composed of three sub-figures (a, b, and c). The sub-figures are vertically placed and depict respectively the semantics-based standards mapping, the reuse assistant, and the specification of families of processes/products/assurance cases. Coherently with the envisioned way forward, which was sketched in D6.1 [17], this vision integrates and extends the results achieved by OPENCOSS and SafeCer projects. This vision incorporates and cross-fertilizes “from-scratch” reuse (reuse, which is not planned/enabled from concurrently engineered assets, part of the same family) and not-from-scratch reuse (reuse, which, instead, is planned/enabled from concurrently engineered assets, part of the same family).

More specifically, on the top of Figure 1 (subfigure (a)), from-scratch reuse is in focus. In particular, semantics-based automatic identification of commonalities is proposed as a solution to identify reuse possibilities from scratch. This solution builds on top of initial explorations, conducted in the framework of SafeCer and in this document empowered by considering recent advances in the semantic web and in tools interoperability. For sake of clarity, it should be noted that subfigure (a) is taken from [87] and was the underlying approach that was already extensively recalled in D5.1 [14].

Once commonalities are identified, the reuse assistant (sub-figure (b)) is expected to exploit them in order to perform a more powerful compliance gap analysis.

Moving on to the bottom of the Figure (subfigure (c)), systematic reuse is in focus. In this case, reuse is not expected to be done from scratch or ad-hoc. Gathered experience is here systematized. Gathered experience is expected to be derived from the left side of the figure. Systematization is conducted by properly engineering the domain and then by deriving desired processes/products/assurance cases via valid configuration. For sake of clarity, it should be noted that subfigure (a) is taken from [86] and was already included and extensively explained in D6.1 [17].

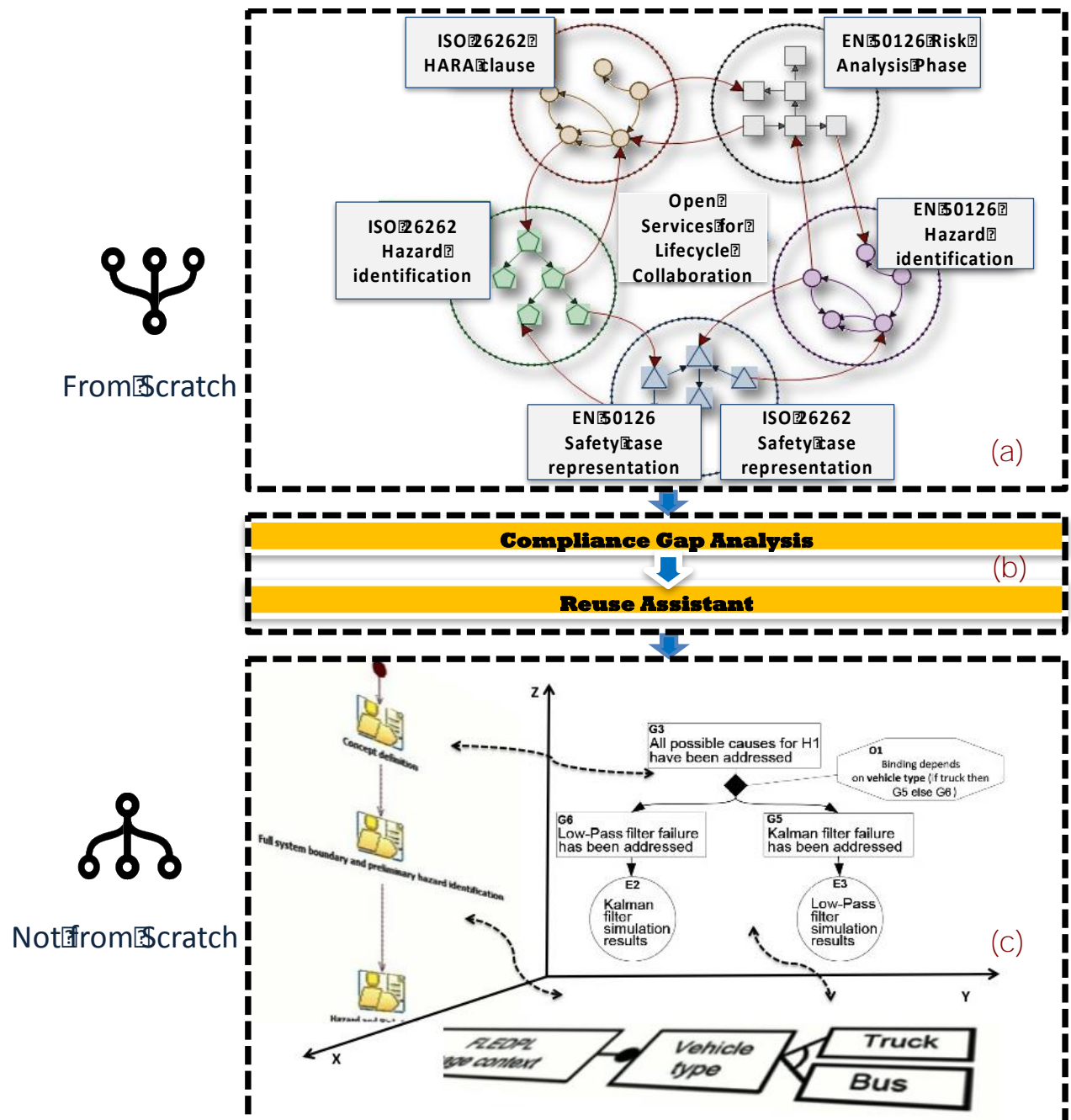


Figure 1. AMASS Vision for Cross/Intra Domain Reuse





## 4. Conceptual solution

This chapter addresses the conceptual solution for intra and cross-domain reuse. To realise the vision presented in Chapter 3, first of all it is necessary to identify which candidate reusable elements should be targeted and at which granularity. As it was recalled in Chapter 2, various reuse scenarios are of interest in the context of the AMASS Use Cases. These scenarios embrace three coarse grained macro elements: process, product and assurance case.

However, the monolithic reuse of these coarse-grained macro elements is not feasible. The internal structure of these elements needs to be revealed in order to identify more fine-grained reusable micro elements. Once the macro and micro elements are conceptually identified, the intended AMASS conceptual and design solutions, aimed at targeting reuse of specific elements or structures, are proposed.

### 4.1 Process-related reuse

This section explains the concepts and conceptual solutions that are needed to implement the cross and intra domain reuse-related functionalities (Prototype P1 and P2), when process-related information is in focus.

#### 4.1.1 Process-related macro and micro (reusable) elements (\*)

To enable reuse of process-related information, first of all macro and micro reusable elements need to be identified. Typically, a basic process structure is constituted of: a unit of work, (a set of) role(s), (a set of) work product(s), (a set of) tool(s), and (a set of) guideline(s).

- Unit of work: indicates what should be done.
- Role: indicates who is responsible for the execution of the work.
- Work product: indicates the documents, more generally artefacts, which are expected to be produced during the execution of the work or used as input information to be able to execute the work.
- Tool: indicates the application that should be used to automate/support the execution of the work.
- Guideline: indicates specific guidance, methods and principles that should be followed during the execution of the work.

All these constituting elements can be considered as reusable micro elements. The above-given basic process structure can be used to create more complex process structures: by chaining and/or by nesting. When an entire basic structure or a more complex process structure can be reused, we are already addressing reusable macro elements, called *process patterns*.

Note that the above listed concepts are part of the most widely used languages for process modelling. In process engineering-related literature, various reference life-cycle models have been proposed for developing systems (e.g., waterfall, V-model, etc.). In the context of safety-critical CPSs, the V-model is the one that is frequently suggested within e.g. automotive standards.

Note that in the context of certification/self assessment it is fundamental to be able to model processes that represent plans (as safety plans) as well as processes that represent the actual execution of the plans. In ISO 26262, Part 2, Clause 5.2.2, for instance, it is stated that “The key management tasks are to plan, coordinate and track the activities related to functional safety”. The result of planning is a safety plan. The result of coordination and tracking is the model of the executed plan (where the deviations are also highlighted).

More specifically, in ISO 26262, Part 2, Clause 5.4.2.2, it is also stated that the organization shall institute, execute and maintain organization-specific rules and processes to comply with the requirements of ISO 26262.

Moreover, these models (the model of the plan and the model of the execution) should be tailored according to the criticality level of the system under development as well as according to the recommended safety, performance, protection or security level. Thus, these levels represent additional concepts that need to be modelled to support the tailoring. In addition, since different domains have different notions for these levels,



ways to compare them, when possible, should also be considered.

Given this tailoring possibility, it becomes clear that a single process model does not fit all development needs. One size does not fit all. An entire family of processes (process line) is embraced. Thus, additional concepts are needed to enable the systematization of reusable process elements between family members. Such concepts are:

- Process-related commonality: indicating the process elements that do not vary and that characterize the family of processes.
- Process-related variability: indicating the process elements that vary and that characterize the individuals within a family of processes.
- Process-related variation point: indicating points of variation where a process element may represent:
  - Process-related options
  - Process-related alternatives

Families of processes in the context of safety-critical systems engineering are called:

- Safety-oriented Process Line (SoPL) [53], when families are characterized by the processes derived from safety-related standards.
- Security-informed Safety-oriented Process Line (SiSoPL) [52], when families are characterized by the processes derived from safety-related and security-related standards, more generally, from multi-concern standards.

The above-listed and discussed process-related macro and micro elements can be easily identified within the current certification frameworks. It is however well known that there are ongoing initiatives e.g., in the US, promoted by Federal Aviation Administration (FAA), as well as in Europe, developed within the RESSAC (Re-Engineering and Streamlining the Standards for Avionics Certification) project, aimed at streamlining avionics certification.

From what has been published so far [163], the goal of these initiatives is to move back to fundamentals and identify a set of overarching properties. These identified properties are:

1. Intent – The defined intended behavior is correct and complete with respect to the desired behaviour.
2. Correctness – The implementation is correct with respect to its defined intended behavior, under foreseeable operating conditions.
3. Acceptability – Any part of the implementation that is not required by the defined intended behavior has no unacceptable safety impact.

Given these properties, which in turn are given at conceptual level and applicable at system/subsystem level, the certification process to show them becomes less time consuming. However, also within this new approach, a planning phase is defined and processes need to be defined and the executed.

More specifically, the requirements for processes that have been elaborated and documented in the initial published results are:

- Identifies the aim of the process.
- Identifies the type of evidence to be produced.
- Defines the means of performing the process.
- Defines any limitations.
- Identifies the environment to be used for the process.
- Requires the identification of the artefacts used in the process be recorded.
- Requires the identification of any additional artefacts used for supporting the processes be recorded.
- Defines any additional constraints that should be satisfied to perform the process.

Thus, the AMASS approach, conceived within this chapter and designed in the next one, remains valid.



## 4.1.2 Summary of previously conceived and validated conceptual solutions

Deliverable D2.2 [4], as well as its latest update D2.4 [6], contains the documentation related to the design solutions for the management of process and standards' information, including mapping between processes and standards. To model processes representing plans, an UMA (Unified Method Architecture) meta-model-based solution was proposed. It should be recalled, for sake of clarity, that UMA was initially introduced as an evolution of the OMG (Object Management Group)'s SPEM 1.1 [116] specification and then it further evolved to offer a certain coverage of the OMG's SPEM2.0 [117] specification.

Figure 2 recalls the UMA-based solution that integrates the concepts presented in Section 4.1.1. In Figure 2, the meta-classes representing roles, work products, units of work (e.g., tasks), as well as process patterns, called in UMA as Capability patterns, can be easily recognized.

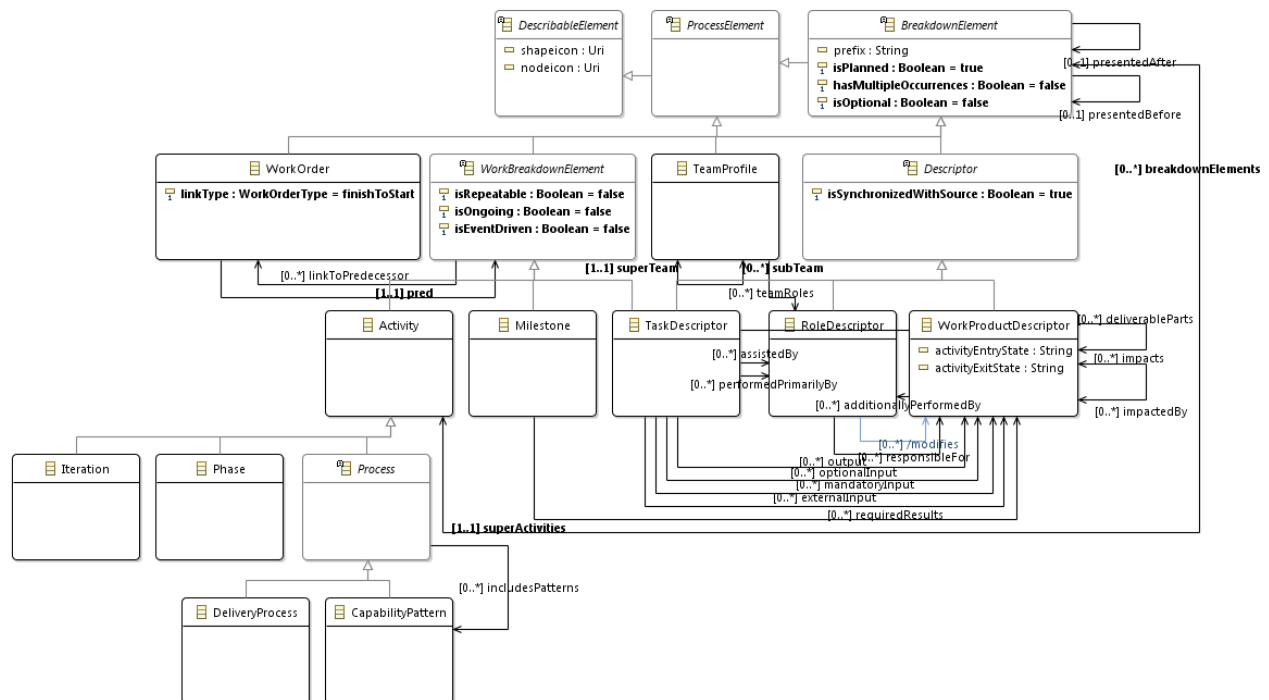


Figure 2. UMA-based solution partly supporting process reuse

To model the processes representing executed processes, a refactoring of CCL (Common Certification Language), result of OPENCOS, was proposed and developed within WP5. From the validation phase, documented in D2.6 [5], it emerged that the modelling of the micro/macro reusable elements is satisfactory.

UMA however does not offer a flexible and powerful support for commonality and variability modelling.

## 4.2 Product-related reuse

This section first provides a recap regarding the challenges and needs associated to product reuse, and then it explains the concepts and conceptual solutions that are needed to implement the cross and intra domain reuse-related functionalities (Prototype P1 and P2), when product-related information is in focus.

### 4.2.1 Recap on challenges related to product reuse

#### 4.2.1.1 Out of context/In-context

Product related reuse strives for building a component once and re-use it in different applications or products. Since the scope of reuse is sometimes difficult to define, we speak in general of reusable assets, or, when it comes to model-based design, of (model) elements. The elements may represent whole subsystems, i.e.,



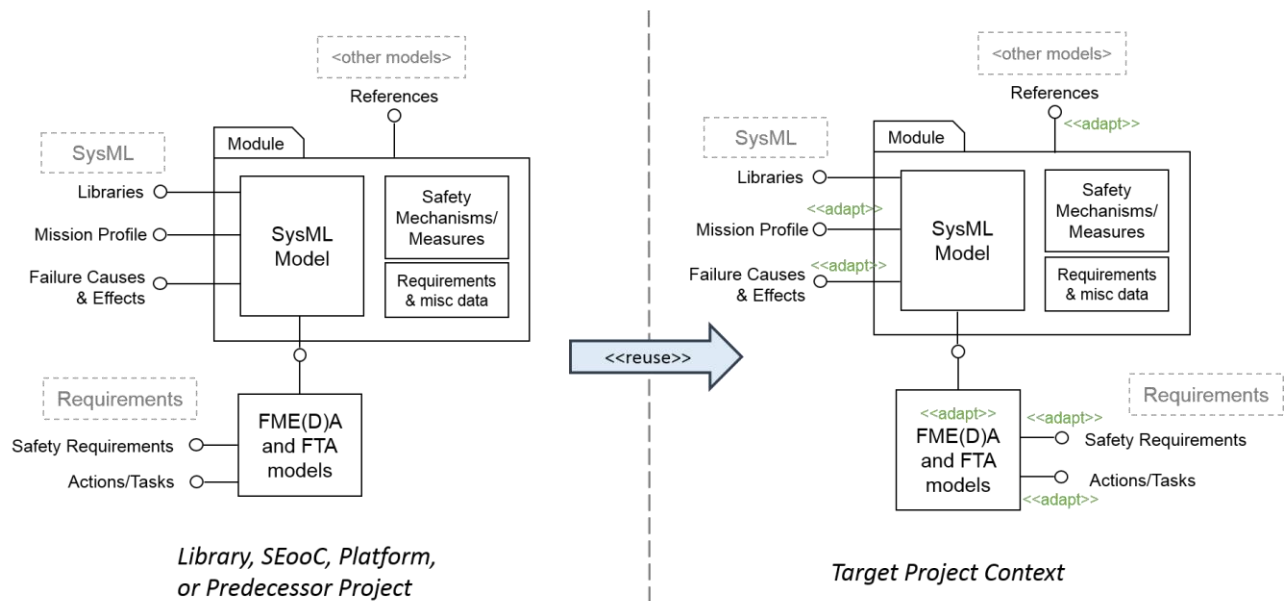
composite components (HW, SW, both), or just parts of them (e.g. atomic components or components with lowered functionalities), for example a SW component with only a subset of its functional capabilities. Together with these design model elements a set of accompanying documentation is exchanged and reused, e.g. (safety/security) requirements or certain parts of safety analysis such as FMEDA (Failure Modes, Effects and Diagnostic Analysis) or FTA (Fault Tree Analysis) that analyse the failure behaviour and put it in relation to a (sub-)system context.

In order to reuse a component, which has been designed, developed, and analysed with respect to functional safety, an elaboration on these (system) *context dependencies* of the component itself is required. These dependencies can be seen as *interfaces* and need to be taken into account to define a clear safety contract, which is then reflected in the safety analyses. For example, component failure modes need to be re-visited and their effects and causes can be assessed with various analyses such as FMEDA or FTA.

The dependencies of a component are multi-fold and exist on various levels regarding the development process and the product context, for example:

- A hardware component has electrical and physical interfaces to the circuit it is integrated in.
- Software components have multiple interfaces to other SW components, but also to the implementation platform (e.g. middleware, operating system, HW dependencies for memory and interrupts, etc.).
- Components consisting of hardware and software might have configuration options to work in various environments. Examples include microcontrollers with fixed HW interfaces but configurable software/firmware/libraries, FPGA (Field Programmable Gate Array)s that can be adapted almost freely to new contexts, complete ECU (Electronic Control Unit)s that support configurable software and patching, etc.
- Physical stress (vibrations, mounting) as well as thermal energy flow, unwanted electrical interferences or EMC disturbances between components and/or the physical/electrical environment are additional interfaces that might be considered with respect to safety.
- The configuration on how a component is (re-)used is of importance for putting it into a new context, e.g. which safety mechanisms are activated for use.

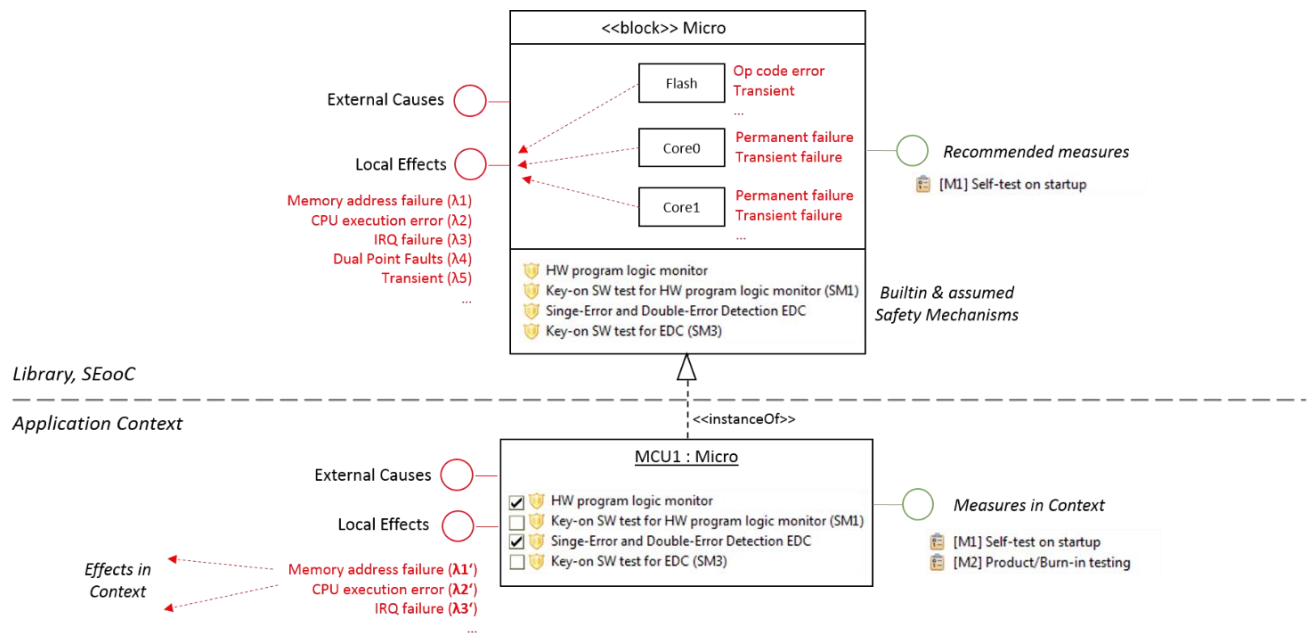
Allocated requirements – especially for safety and security – are the most important interface/dependency of the component model that needs to be considered since they define the performance and characteristics of a component. These design level, also known as model information, provides also the connection to the safety, security, and simulation analyses.



**Figure 3.** From out of context to in-context: the challenge of component reuse

Based on these considerations, reusing a model of a component requires also to reuse the associated analysis artefacts such as FMEDA, FTA, Requirements, and others. These *analysis models* must be taken into the reuse context and they need to be adapted, as shown in Figure 3. A component has all the interfaces/dependencies described above (here depicted using the typical UML notation), which must be adapted to the target application context. The figure shows for example a component model expressed in the SysML modelling language and its related model information such as used libraries, mission profile with stress parameters, like temperature, lifetime specification, maintenance, etc., relevant for failure rate determination, potential failure causes and effects from/to connected components, and so on. The reusable component can be seen as a “module” which comes as a model (i.e. the component definition itself) with a set of additional information (e.g. configuration options, built-in safety mechanisms, safety requirements, etc.) which can be seen as meta-data about the component itself. We use the term *module* because of lack of a better wording for this set of related data. However, most of this data has to be processed by analysis (manual, automated) during the integration of the component into the reuse context.

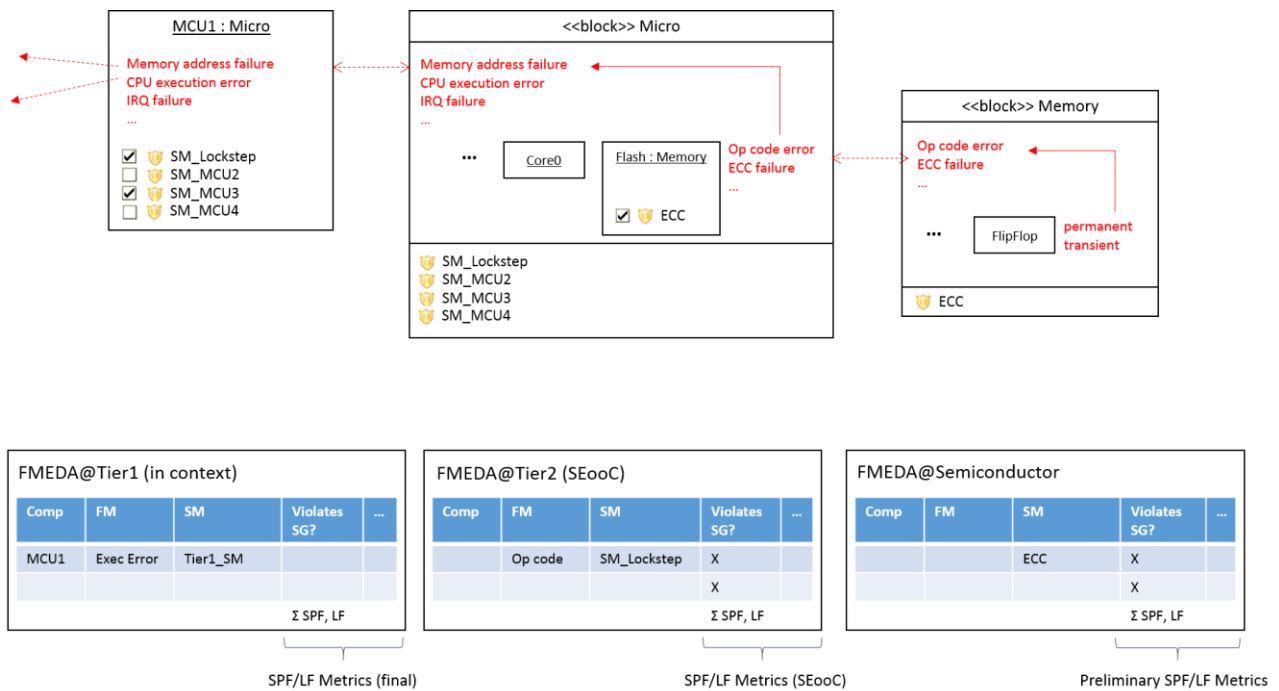
At the modelling level of a component, a formalization of the safety characteristics is required to enable reuse. One formalization approach consists of the safety contracts described in D3.1 [8], D3.2 [9], and D3.3 [10] deliverables of AMASS. Besides the compatibility that can be expressed there for runtime verification of safety properties, the fault model is a design level model extension that is required to perform reliable and consistent analysis for a reused component. One aspect here is the possible *failure modes* of a component and the *safety mechanisms* which are built into the component or assumed from the context. This qualitative information, together with the failure rate and use assumptions, is often a minimal set of required information for validating how a component is affecting the context it is embedded into. Figure 4 shows some more details about these implied interfaces of a component:



**Figure 4.** From out of context to in-context: focus on the interfaces

For example, a microcontroller that is reused might have been analysed and its failure characteristics might have been assessed as a Safety Element out of Context (SEooC) or in a predecessor project context (“proven-in-use”, same domain or different application domain). The analysis results can be represented as an extension to the component models interface, failure properties depicted in red, safety mechanism and measures in yellow and green. When reusing such an assessed component all this safety-related information has to be adapted (i.e. connected) to the new context. This step has to be supported by the modelling capabilities and the tools in an (semi-)automated manner.

The situation gets even more important when taking a deeper look at reuse and integration chains, e.g. for a Tier 1 in the automotive domain. While the Tier 1 is an integrator for a set of hardware components developing its own software, a lot of the hardware components are reused and the demand is there to reuse and adapt the safety analyses of these components. The semiconductor companies supplying the HW parts are also integrators of a number of (reusable) IP design information. Therefore, reuse is challenged along this integration hierarchy to have a consistent set of analyses for the final safety case. Figure 5 shows an example of an FMEDA hierarchy for this example case:

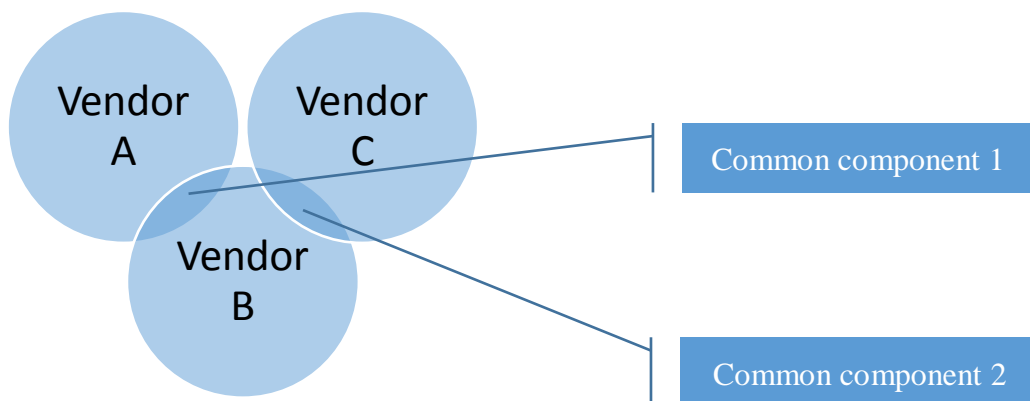


**Figure 5.** FMEDA hierarchy within the automotive domain

Each analysis level takes input data that comes ideally with the component model which is reused. An IP provider might deliver a set of FMEDA (or at least failure modes/failure causes analysis) to the semiconductor manufacturer, who in turn aggregates the chip level information for use in the Tier 1 context. While the first two analyses at IP and semiconductor level might have incomplete information from the system context, the Tier 1 might be able to complete the analysis, taking into consideration all the various configuration options and environmental constraints.

#### 4.2.1.2 Common components

The usage of common components represents a challenge. For example, there are several different function implementations from different vendors, and they are reused from an open-source library:



**Figure 6.** Conceptual overview of common components from a vendor's perspective

- Common component full internal access of the source code (total openness)
- Common component developed by third-party (closed)
- Common component developed as open-source (shared development)



In case of development, where total control of both products and processes exists, total openness of the evolution of the product and its quality control process is fully manageable. In case of third-party involvement, the component is more or less a black-box in this sense. The third case is relevant where the product's internal design is visible (source-code, etc.) but the process and strategy of the development is under control of a third-party.

If there is full access to both the development process and its result, it is possible to address safety assurance according to classic methods and tools. Using the AMASS platform may then focus on lowering the needed effort and improving the quality of the assessment work. Even though there is full visibility in the development, a large system project may have separated teams and software components. This may imply that the size of the project will generate a more complex situation similar to the case of external ownerships of the component.

#### 4.2.1.3 Safety/Security Analyses Reuse together with component reuse

The reuse of the model of a component demands the reuse of the associated analysis artefacts. As long as the component model introduces product-reuse aspects (configuration options, environmental constraints, etc.), the analysis artefacts should also introduce built-in features to fit in. To some extent and for certain domains, the coupling between safety and security is nowadays better understood. For instance, the European standards ED-202A [101] and ED203 [103] provide guidelines and methodological support to integrate security and safety in the development cycle of aeronautics systems. Industry and academy recognize that a joint analysis is required to improve product confidence. This is particularly true for systems that were originally designed and deployed to operate locally and mainly/only considering safety aspects. For instance, in the case of Industrial Control Systems (ICS), security has progressively become a concern as long as ICS connectivity increased, and new threats emerged. In addition, the evolution in systems engineering shows that, in the past, safety and security analyses had been proposed and conducted independently and the need for a joint safety-security analysis is relatively recent. Several efforts have been already conducted and implemented to address this concern, e.g., IEC 63069 for industrial process automation [104]. Other similar initiatives, like ISO 26262 Edition 2 FDIS for the automotive domain, are currently in progress and their results are expected to be released soon. In this context, the identification of commonalities and variabilities between safety and security concepts and methods is an important work to achieve the joint consideration of safety-security aspects from early stages of the design process. To render this consistent, the following aspects are considered:

- *Regulation*: the government policies, international or domestic standards, technical recommendations, requirements, etc., issued to ensure or improve safety, security and, more recently, safety-security.
- *Methodologies*: the works performed by industry and academia to propose methods to conduct safety, security and safety-security analyses.
- *Knowledge bases*: the definition and evolution of knowledge bases that contain the concepts, categories, types, patterns, etc., useful to specialize safety and security aspects, for instance, according to the product domain or for cross-domain use.
- *Frameworks and tool support*: the development of languages, frameworks and tools to support and automate as much as possible the analysis methods. Tool development is also concerned with the integration of knowledge bases. To ensure seamless product reuse from design, a certain level of tools/frameworks interoperability needs to be achieved.

The following items describe relevant variability aspects to be considered when targeting reusable elements amenable for security and safety analyses:

- *Analysis criteria*: the typical safety criteria are related to product reliability, failure rate, and robustness against natural or accidental use and failures. The typical security criteria are related to product/system/data/exchanges integrity, availability, confidentiality, authenticity, freshness, non-repudiation, controlled access, and privacy, within a hostile environment, intentionally exploiting vulnerabilities by attacks.



- *Analysis goals:* The safety and security goals are settled to ensure product/system trustworthiness with respect to the associated criteria. The goals accomplishment depends upon the efficacy to elicit appropriate requirements and their dependencies. In the context of AMASS, a joint analysis driven by safety is proposed. It means that a security criterion is introduced as long as an impact on safety is identified. Thus, the security criteria having negligible or no effect on safety are left out. The initial security criteria that likely impact safety, in case of non-authorized access, are:
  - Integrity
  - Availability
  - Confidentiality
- *Product context:* in a safety analysis, the context is often fully mapped into the component/system model (hazardous events, failure conditions and propagations, component reliability, etc.). The mapping finally yields classical outputs like Failure Mode, Effects and (Criticality) Analysis FME(C)A tables, FTA diagrams, etc. The context in a security analysis is hostile, independent, and smart. An attacker model is independent from the safety analysis, and it is modelled relying on several structures like Attack Trees, Attack-Defence Trees, Threat Scenarios, Misuse Scenarios, and severity of impact and efforts required for a successful attack, etc.
- *Evaluation metrics:* the risk is a metric used to evaluate both security and safety, but different for safety and security context (IEC 61508 vs. IEC 62443). The risk (security) is measured in terms of the likelihood of failures/attacks occurrence (which is not necessarily a probabilistic value in the mathematical sense) and the severity of consequences (which is sometimes not known). In safety, the risk is defined as combination of the probability of occurrence of harm (caused by a failure, therefore the SIL are probabilistic terms) and severity of that harm [105]. In security, several metrics have been proposed and used to evaluate risks. Indeed, whereas severity is commonly accepted and used to evaluate attacks impact, several qualitative and quantitative metrics have been proposed to evaluate attacks likelihood. As for qualitative metrics, we can mention the resources, skills, and complexity for attack preparation and accomplishment. As for quantitative metrics, there exist some efforts to reuse probability distributions so as to determine attack probability, e.g., [106], [107]. Even if some standards like IEC 62443 [109] introduce principles for quantitative risk assessment, the existing security methods still lack of quantitative metrics to adequately evaluate aspects like attacks difficulty, probability of attack actions/paths, and realistic times for attack occurrence, and consequences, which may not be only physical harm, but indirect operational harm, loss of property, etc. In addition, the evaluation of countermeasures efficacy is conducted against an attacker model. The definition and implementation of an attacker model should consider attacker categories, motivations, resources, capabilities to search and exploit vulnerabilities, and variability. Building such model often demands the integration of several methods, frameworks, and techniques.

#### 4.2.1.4 Verification phases of the life-cycle maintenance of SEooC (\*)

In the context of ISO 26262, verification of safety critical parts (focus on software elements out of context (SEooC)) constitutes an important part of the product development project effort. Such effort is expected to increase in the case of safety-critical parts that can be affected by security-related aspects. Cross-concern reuse of verification results represents a challenge.

With the rapid growth of software in CPSs, there is a need to automate the verification of software as well as enable reuse of verification results. The traditional model-based system engineering (MBSE) methods and more specifically the available model-based verification methods do not seem to be adequate to enable (cross-concern) reuse.

The challenges of software SEooC for high functional safety (and cybersecurity) systems are related to:

- The maintenance and regressions tests due to the needed efforts to keep the modelling in synchronization with the software. This effort is needed in many development projects, where no





tight automatic connections exist between the system model and the code (i.e., no code generator is implemented).

- The divergence in synchronization between the system models and the code, when automatically generated code is patched.
- The test re-execution, which might be needed to fulfil ISO 26262 requirements to guarantee the expected code coverage.
- The insufficiency of MBSE with manual inspections of models, when high-level safety integrity is expected.
- To provide the inputs for software verification there needs to be exhaustive test drivers, as the SEooC may have many different inputs in the different contexts.
- The reuse of the component requires that the components have been exhaustively verified for all possible combinations of inputs.

Software testing generally consumes between 30 and 60 percent [146] of the overall development, therefore it is of vital importance to address the testing aspect when developing the methods and tools for re-use in CPSs:

In the waterfall development cycle, as referred in both ISO 26262 and IEC 61508, there is a large emphasis on the right (ascending) side of the V-model in said waterfall development model.

WP6 is addressing the methods and tools for re-use in CPSs. The need for efficient testing is of vital importance, due to the following reasoning:

- In large systems there will be a mixture of re-used and non-reused software components.
- When addressing a complex CPS, with possibly multi-concern, we need many test cases.
- The automated model-based testing (aMBT) is one of the strongest solutions to address this challenge that grows with the complexity of the CPS.

Due to the complexity of testing, which is demanded by the highest integrity levels within the IEC 61508 and ISO 26262 safety standards, the verification represents a significant effort of the total effort of CPSs development.

For the higher ASIL (in ISO 26262), the verification phase is necessary to the complete system test according to the current standards. The updating of the ISO 26262 standard in 2018 is expected to elaborate on the software cycle but still the verifications phase will be a cornerstone in the product assurance.

The verification phase for the development and maintenance of complex cyber-physical systems (CPSs) for safety critical applications can imply growing costs. In ISO 26262, the V-model for development has strict requirements on the verifications for the safety-critical system – ISO 26262. To enable cost-reduction and reduced time to market for safety-critical CPSs, there is a need for methods and tools for the regression tests to support the intra-domain and cross-domain re-use of software components.

## 4.2.2 Product-related macro and micro (reusable) elements

To cope with the previously recalled challenges that hinder reuse, first of all macro and micro reusable elements need to be identified. Typically, a basic architectural design is constituted of the following elements: components (or blocks) including ports, contracts (classifiable e.g., into weak and strong), and connectors. These elements constitute the micro reusable elements.

When micro elements are composed to build complex and reusable architectural models, such models are called architectural patterns. Micro and macro elements were extensively explained in D3.2 [9].

Similar to processes, these elements may vary in critical systems based on their criticality. For instance, a component may provide a specific service for the highest criticality level and no such service for lower criticality levels. Given this tailoring possibility, it becomes clear that a single component model does not fit all development needs. One size does not fit all. An entire family of products (product line, more specifically



design specification line) is embraced. Thus, additional concepts are needed to enable the systematization of reusable architectural elements between family members:

- Product-related commonality: indicates the product elements that do not vary and that characterize the family of products.
- Product-related variability: indicates the product elements that vary and that characterize the individuals within a family of products.
- Product-related variation point: indicates points of variation where a product element may represent:
  - Product-related options
  - Product-related alternatives

### 4.2.3 Summary of previously conceived and validated conceptual solutions

Deliverable D2.2 [4], as well as its latest update D2.4 [6], contains the documentation related to the implementation solutions for the management of system-and-component related information, including modelling and reuse of architectural specifications. To model and reuse system-and-components (as well as their associated evidence for assurance purposes), a domain specific language, conceived for modelling contract-based component-based systems, was proposed in D3.2 [9]. Figure 7 recalls a subset of the meta-model of such domain-specific language. In particular, Figure 7 emphasises the concepts presented in Section 4.2.2.

CHESSML [143], which sometimes is spelled CHESS-ML in the literature, is a modelling language compatible with such domain specific language.

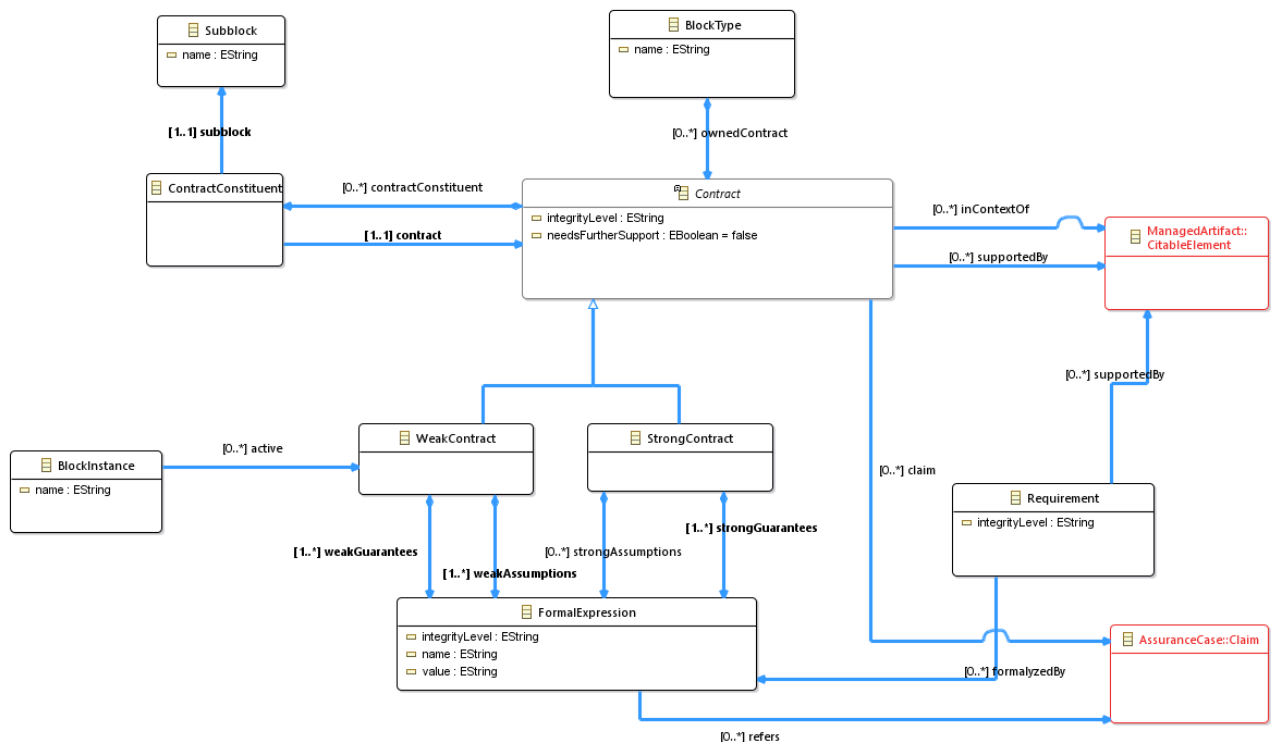


Figure 7. ChessML-based solution partly supporting component reuse

## 4.3 Assurance case-related reuse

This section explains the concepts and the conceptual solutions that are needed to implement the cross and intra domain reuse-related functionalities (Prototype P1), when assurance case-related information is in focus.

### 4.3.1 Assurance case-related macro and micro (reusable) elements

As known, an assurance case is constituted of claims, contextual information, evidence, and reasoning structures aimed at explaining why the claims are sufficiently supported by the evidence. Knowing that the concepts for *Evidence-reuse* are already covered in previous sections (see 4.1 and 4.2), those macro and micro reusable elements targeting reuse of arguments are identified in this subsection. These elements are:

- Module: self-contained, weakly coupled argumentation element.
- Reasoning structures: patterns; the concept of a safety (more broadly assurance) case patterns represents “a means of documenting and reusing successful safety argument structures” (i.e., goal structures in GSN terms).

Similar to processes and products, in critical systems, these elements may vary based on the criticality. Given this tailoring possibility, it becomes clear that a single assurance case model does not fit all assurance needs. One size does not fit all. An entire family of assurance cases is embraced. Thus, additional concepts are needed to enable the systematization of reusable assurance-case-related modelling elements between family members.

- Assurance case-related commonality: indicates the assurance case elements that do not vary and that characterize the family of assurance cases.
- Assurance case-related variability: indicates the assurance case elements that vary and that characterize the individuals within a family of assurance cases.
- Assurance case-related variation point: indicates points of variation where a product element may represent:
  - Assurance case-related options, when for instance an additional branch aimed at developing the argument is not always needed due to optional requirements.
  - Assurance case-related alternatives, when for instance alternative branches aimed at developing the argument can be chosen, due to requirements that can be met in different ways.
- Variability: Two kinds of variability might be identified within a set of assurance cases:
  - Intrinsic: whenever there is more than one argumentation style to support the claims of a particular product line instance (see, for instance, alternative)
  - Extrinsic: whenever reusable assets (referenced in the assurance case and bound to concrete assets within product-line models, such as the feature and reference architectural models) vary.

Remark: Commonality as well as variabilities are both supported in GSN, but only in GSN.

### 4.3.2 Summary of previously conceived and validated conceptual solutions

Deliverable D2.2 [4] contains the documentation related to the implementation solutions for the management of assurance case related information, including modelling of argumentation-related architectures. To model and reuse assurance cases, a SACM-based solution was proposed. Figure 8 recalls the SACM-based solution that integrates the concepts presented in Section 4.3.1.

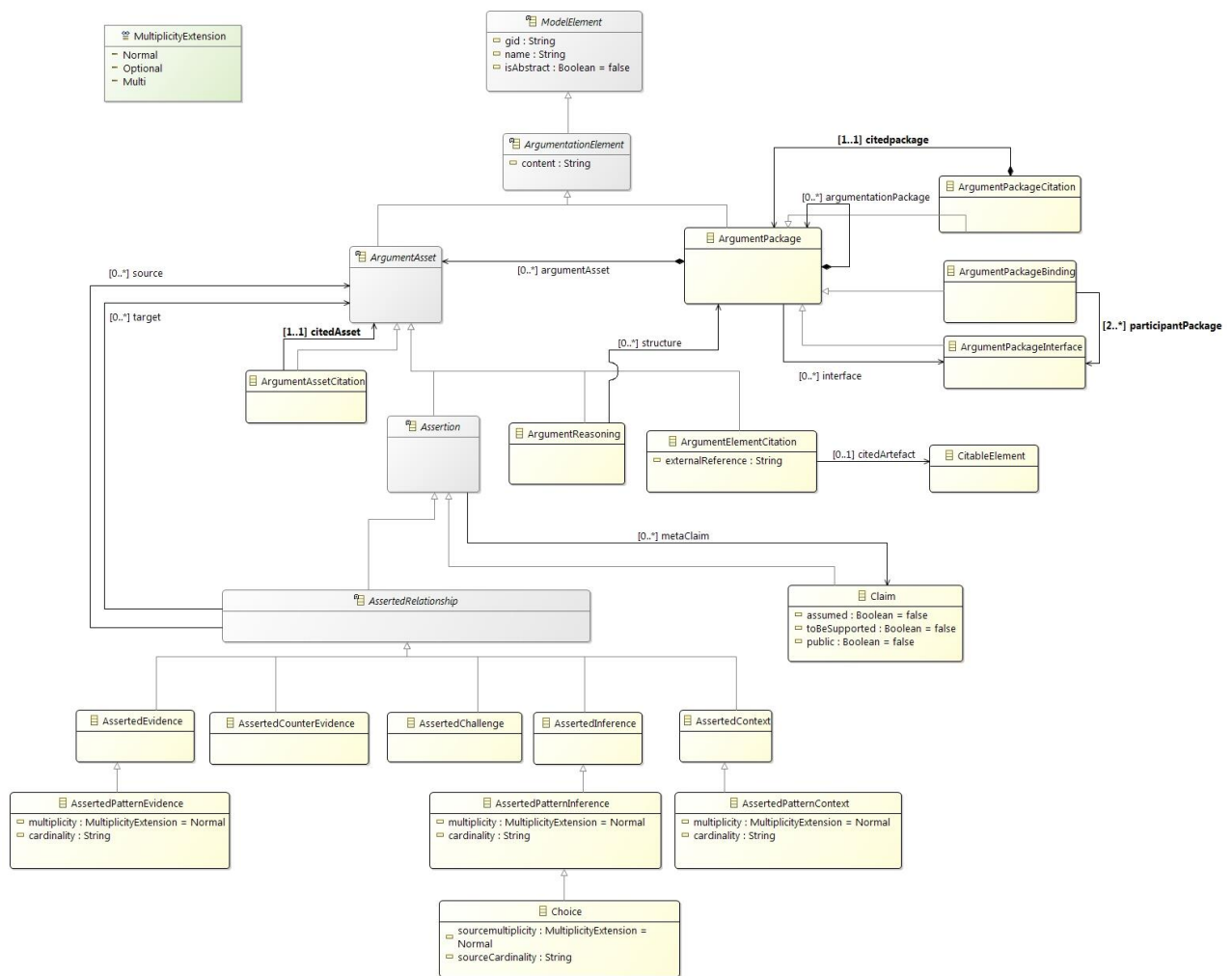


Figure 8. SACM-based assurance case metamodel

Figure 8, however, does not include meta-classes for modelling the management of commonality and variability.



## 5. Design Level Solution

In this chapter, we design the AMASS solution for cross and intra domain reuse. Our solution comprises three main functionalities: the reuse discovery (explained in Section 5.1), the reuse assistance (explained in Section 5.2), and the management of families of processes, products and assurance cases (explained in Section 5.3), including analysis of change impact on individual members of one family, due to changes in individual members of other families. In addition to these main functionalities, other functionalities are part of our global solution, i.e., functionalities enabling reuse of product-related artefacts (Sections 4÷7), change impact analysis via elastic search methods (Section 5.8), as well as functionalities for automatic generation of process as well as product-based arguments (Section 9÷10).

### 5.1 Reuse discovery

In the context of software engineering, reuse [119] [120] [124] is commonly defined as a process to systematically specify, produce, classify, retrieve and adapt software artefacts for the purpose of using them in a development process. In general, software reuse [120] [123] may have the potential of increasing productivity of engineers, improve quality and create a cost-efficient development environment. However, both technical and non-technical issues for a limited reuse can be found [121] [122]: 1) economical, organizational, educational or psychological issues; and 2) lack of standards to represent artefacts, and lack of reusable component libraries or appropriate tools for boosting reuse and interoperability among tools.

In the context of technical issues, systems and software engineering techniques have been widely studied to support the classical principles of reuse [120] [125]: abstraction, selection, specialization and integration. More specifically, abstraction (i.e. management of the intellectual complexity of a software artefact) can be considered the essential feature for any reuse technique to specify when an artefact could be reused and how to reuse it. Selection refers to the discovery of software artefacts, covering from the representation and storage to the classification, location and comparison. Specialization consists of the set of parameters and transformations required to reuse a software artefact, while integration refers to the capability of software systems to communicate, collaborate and exchange data.

Thus, the reusability factor [124] [125] of system artefacts will directly depend on their abstract description, on how they can be selected and specialized for reuse, and how they will integrate into a new software system. Furthermore, a reuse approach implies that every artefact generated during the development lifecycle is not any more an isolated requirement specification, model, piece of source code or test case, but a knowledge and organizational asset. However, after a long time, reuse promises [128] [129] are still far from reaching the major objective of optimizing the system development lifecycle efforts [126], even though tools, techniques, methods and languages and the overall understanding of a system have dramatically changed since the NATO Software Engineering Conference in 1968.

The emergence of Model-based Systems Engineering (MBSE<sup>7</sup>) as a complete methodology facilitates addressing the challenge of unifying the techniques, methods and tools to support the whole specification process of a system, including conceptual design, system requirements, design, analysis, verification or validation. In the context of the well-known V lifecycle model, it means that there is “formalized application of modelling” [131] to support the left-hand side of this system lifecycle implying that any process, task or activity will generate different system artefacts but all of them are represented as a model. This approach is considered a cornerstone for the improvement of the current practice in Systems Engineering since it is expected to cover multiple modelling domains, to provide better results in terms of quality and productivity, lower risks and, in general, to support the concept of continuous and collaborative engineering easing the interaction and communication between people (engineers, project managers, quality managers). MBSE and reuse is currently under study [132] [133] in which component models [130] are applied to enable reuse in MBSE. Furthermore,

---

<sup>7</sup> INCOSE, “Systems Engineering Vision 2020,” INCOSE, Technical INCOSE-TP-2004-004-02, 2004.

variability management techniques are also being explored [134] to link MBSE to product lines and, thus, support the principles of specialization and integration.

However, abstraction and selection processes are not fully developed. Currently, existing interoperability initiatives (such as ISO 10303-STEP or OASIS OSLC) are trying to boost reuse through data exchange or referencing (linking), but there is much more at stake than the mere exchange of data. The first step to be able to exchange (and reuse) data, information, and knowledge lies on the provision of a proper environment for system artefacts retrieval that will be the first step to look up artefacts to finally exchange (and reuse) them through the protocols and common data models mentioned before.

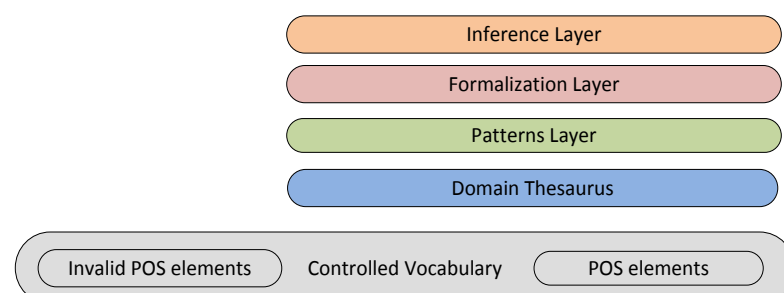
Existing platforms for the management of system engineering processes such as the Jazz Platform by IBM or Papyrus (Eclipse CDO), offer a kind of central repository in which engineers can upload their systems artefacts and perform tasks such as searching, traceability management [135], etc. These centralized repositories represent artefacts as a set of metadata that are linked to a system artefact (content). Although in some cases, the use of metadata can be useful to look up artefacts by filtering certain properties, it is considered too simple to enable the proper reuse of the knowledge embedded in the system artefacts. As a motivating example, in traditional information retrieval systems (text-based) if we are looking for documents (text) we will express queries as text (or keywords), and the search engine will match documents according to the input query. In any case, in all of them, the representation of information, the queries and the results, are working under the same context: text. The same kind of behaviour can be found in the Google Image search service where we can look up images by entering an image.

If the same principles are applied to the systems engineering discipline, we should be able to represent, index, query, and retrieve any kind of system artefact depending on their type, increasing the reusability factor of previous works. Moreover, and taking into account the plethora of tools, system artefacts and formats, an advanced retrieval system for a MBSE platform should be able to represent, store, and retrieve any kind of artefact by using as input query any kind of system artefact: a requirement, an architectural or a physical model or event, just a text (that can be represented in a structured way).

Thus, information retrieval techniques will equip engineers with a method to support more complex processes that require a holistic view of a system such as the traceability link recovery process as part of the verification and validation technical processes. Some specific works can be also found in this area for retrieving physical system models (e.g. Modelica RC circuits [136] or SysML models [137]). However, the notion of a complete, semantic-based, retrieval system for system artefacts and MBSE platforms is still under development and should enable the first step for a proper reuse environment: search and discovery.

### 5.1.1 Methodology to represent system artefacts

Ontologies are commonly used to model domain knowledge in some area using a particular syntax and logic formalism. Some of the classical definitions [138] [139] describe an ontology as a specification of a conceptualization; that is, as a set of concepts (classes), attributes and relationships aiming to share and reuse knowledge. In the context of requirements authoring, the use of an ontology can help to restrict the concepts that can be used to describe a requirement on all lexical, syntax and semantic/category levels, see Figure 9. In general, a knowledge base built as an ontology can be layered as follows:



**Figure 9.** Layers of an ontology-driven approach to implement a Knowledge-Centric Systems Engineering strategy.

- **Controlled vocabulary layer** contains all those concepts with a specific meaning in a domain.
  - **Part-of-Speech (POS) elements layer** includes all those terms that are part of the speech and are used to build domain-based terminology and concepts such as prepositions, conjunctions, articles, etc.
  - **Invalid POS elements** is the set of terms that should be avoided in textual requirements to reach desirable quality characteristics.
- **Domain thesaurus layer** is comprised of those concepts and terms that have relevance in a domain but include more sophisticated and particular relationships that can be defined either in a metamodel or inherited from a typical thesaurus structure, such as hierarchical relationships (e.g. *broader/narrower*) or composition (e.g. *part-of/whole-part*).
- **Pattern layer** defines the proper grammar to create pattern-based system artefacts. It makes use of the existing definitions (concepts) by exploiting relationships such as the lexical and semantic ones (e.g. synonymy or *part-of*).
- **Formalization layer** is the layer in charge of managing semantic relationships and exploiting the underlying knowledge that has been formalized through concepts and relations.
- **Inference layer** represents the rules that can be used to infer new knowledge based on both the underlying data model (e.g. a semantic graph) and a certain type of logics. In some contexts, such as expert systems, this layer corresponds to the use of a semantic-based reasoner or a rule-based engine.

Some advantages of using formalized knowledge from ontologies in Systems Engineering processes can be highlighted:

- With regards to system artefacts authoring:
  - a. Identify the domain concepts that are participating in some model or diagram.
  - b. Model and automatically process the structure (grammar) of any kind of system artefact based on the restrictions established in a metamodel.
- Formalize, as a semantic graph, any kind of content. Since most of (static and structural) models can be thought of as an underlying graph in which relationships connect nodes of different types, the outcome of a common representation model would be represented by a semantic graph aligning all concepts and relationships under the frame of an ontology. Thus, it is possible to calculate quality metrics that can potentially have impact in the reusability factor of an artefact. Those quality metrics must necessarily cover:
  - a. Knowledge for calculating correctness metrics (artefact level).
  - b. Knowledge for calculating consistency metrics (specification level).
  - c. Knowledge for calculating completeness (artefact and specification level).

Reuse discovery is then designed as a process exploiting the underlying semantics (concepts and relationships) and used to describe the different system artefacts. In this case, the process is based on matching similar underlying graphs (since every piece of knowledge is modelled as a semantic graph).

According to the previous introduction, a reuse discovery process based on the use of ontologies (semantics) is designed as a function in which language descriptions are elevated to a concept-based representation, exploiting the typology of concepts and relationships coming from a metamodel. This functionality can then be defined as a function  $S$  that for a given resource  $r_k^i$ , a target set of resources  $R_j$  and a context  $C$  (containing information about natural language processing such as stop words, acronyms, etc. and metamodels) will generate a set of results  $\{(r_k^i, r_k^j, c)\}$  where the input resource and other resource  $r_k^j$  are matched together under a certain value of confidence  $c$  as the next equation shows:

$$S: r_k^i \times R_j \times C \rightarrow \{(r_k^i, r_k^j, c)\} / r_k^i \in R_i \wedge r_k^j \in R_j \wedge c \in \mathbb{R}$$

This definition can be widespread and applied to the mapping process between two different sets of resources,  $R_i$  and  $R_j$  as the next equation also shows:

$$S: R_i \times R_j \times C \rightarrow \{ (r_k^i, r_k^j, c) \} / r_k^i \in R_i \wedge r_k^j \in R_j \wedge c \in \mathbb{R}$$

Given the two previous definitions, a reuse discovery process based on a semantic search function can be seen as a mapping process in which two artefacts (resources),  $R_i$  and  $R_j$ , are used as source and target sets of resources.  $P$  is the set of patterns (metamodel specification) that have been designed to serve us to design the resources in both specifications using a set of domain vocabularies  $O$ : generally, one ontology will be enough to represent the domain knowledge. The output of this function will be again a set of mappings  $\{ (r_k^i, r_k^j, p_i, p_j, c) \}$  where:

- $r_k^i$  represents a resource in the source specification, written following the pattern  $p_i$ .
- $r_k^j$  represents an artefact in the target specification, written following the pattern  $p_j$ , and
- $c$  is a value of confidence.

Thus, it is possible to discover similar artefacts by performing a matchmaking process based on the semantics of the relationships defined at different levels: lexical (words), syntax (metamodel), and semantics (typology of nodes and relationships).

Although this definition of a reuse discovery process allows us the possibility of elevating the meaning of text-based and metadata resources to a semantic-based representation, the main and common drawback of this approach lies in the necessity of human-validation to ensure that the mapping is 100% correct. However, the possibility of suggesting matching resources by exploiting the semantic relationships in a domain ontology can boost the reusability factor of any system artefact generated during the development lifecycle of a critical system.

### 5.1.2 Architecture and Operations to support reuse discovery

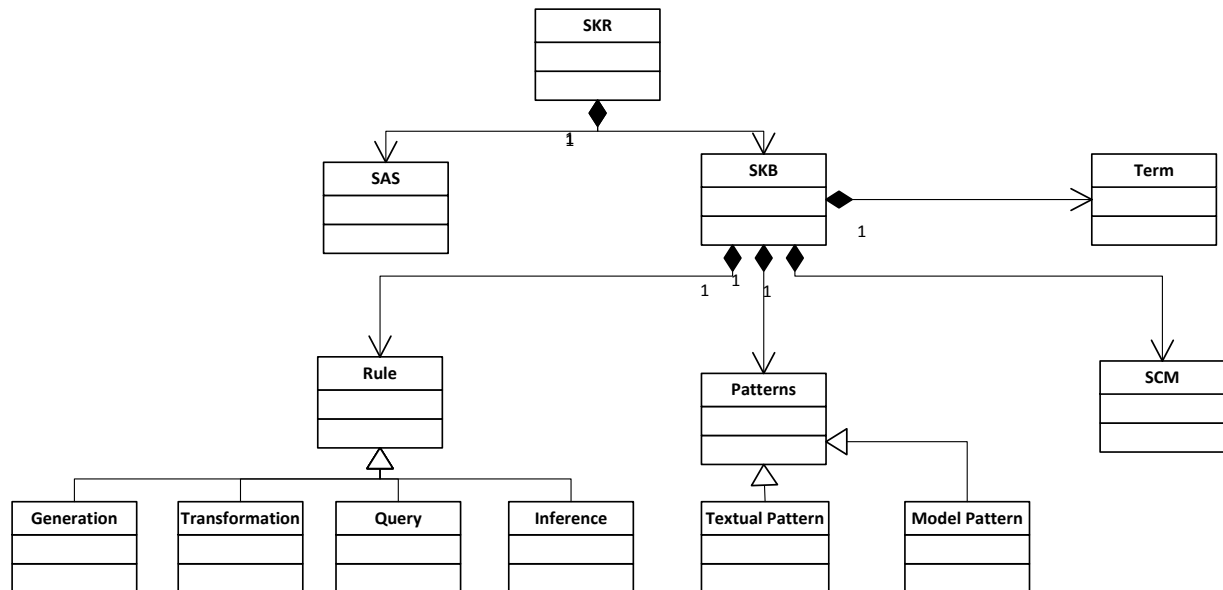
The logical architecture for providing reuse operations is based on the interoperability layer defined in deliverable D5.2 “Design of the AMASS tools and methods for seamless interoperability” [15]. In this context, the following elements must be mentioned:

1. There are two main conceptual blocks:
  - a. The SKB (System Knowledge Base) that contains the “ontology” to drive the process of reuse discovery based on the creation and exploitation of semantic relationships. This block mainly comprises the system conceptual model covering: terminology, taxonomy, patterns and rules, as it was previously outlined.
  - b. The SAS (Systems Assets Store) that contains the formal representation of system artefacts based on the use of the concepts and relationships of the SKB.
2. The set of operations to manage both conceptual blocks provides a service layer based on the extension of the OSLC Concepts “Delegated User Interface” and “Delegated Operation”. In this case, any reuse operation is offered in terms of the OSLC resource shape defined for exchanging knowledge and, following the deliverable D5.2, the resources shape is described as SRL (System Representation Language). The resource shape, see also Figure 11, which provides the schema to define the input/output interface of the reuse operations layer is based on the following class diagram. In this manner, any input content and output artefact must be expressed following the entities represented under this model and creating an underlying graph that is used to implement a search engine to discover potential reusable assets.

Building on the previous principles, a system knowledge repository is created to support the representation, storage, and retrieval of system artefacts as the next figure shows. Here, a System Knowledge Repository (SKR)



is composed of a knowledge base and a realization of this knowledge base through assets that are represented in the SRL resource shape.



**Figure 10.** A system Knowledge Repository structure.

Thus, the building blocks of the architecture comprise two main aspects:

1. The services interoperability layer, where a component implementing the OSLC KM specification (see D5.2 [15]) is offering services to external clients and delegating the real implementation of operations to the CAKE (Computer-Aided Knowledge Environment) API provided within The Reuse Company tools.
2. The CAKE API that provides the implementation of the reuse operations exploiting the information available in the Knowledge Base and the formal representation of the assets that are stored in the repository.

Once the architecture is defined by separating the specification from the implementation, a set of operations are designed to support the main functionalities expected for delivering a reuse discovery mechanism. To do so, Table 3 shows the main operations available to manage the System Knowledge Base.

**Table 3.** Common operations definition for the System Knowledge Base

Operation	Description
Text standardization (Normalization)	When dealing with names of concepts and relationships presented in some model, it is possible to find different lexical variations (e.g. number or gender). It is important to unify such text descriptions to be able to link and compare different artefacts at a conceptual level.
Validation	When dealing with names of concepts and relationships presented in given model, some of the terminology could be out of the domain. This operation ensures that only valid names will be accepted for the purpose of reuse.
Preferred	When having a term, there is a preferred term for that concept.
Related	When having a term, concept or even a system artefact, a set of related resources can be found by simply querying the metadata of such resource.

More specifically, a brief description of each operation is provided below:

- Core operations: Create, Retrieve, Update, and Delete operations for the elements comprising the SKB.



- Common operations: set of operations aimed at providing the unification and standardization of the terminology that can be found in any asset.
- Retrieval operations: this mainly covers two functionalities for indexing and searching any kind of artefact that is part of the SKB. In this case, elements of the SKB are also considered assets.
- Reuse operations: this set of operations covers the management of the SKB being able to copy one knowledge base into another, merge two different knowledge bases and perform a delta operation (DIFF) between two different knowledge bases.

**Table 4.** 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 Standar- dization	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	

### 5.1.3 Design of a research method to evaluate a reuse discovery process

To illustrate the approach for reuse discovery, a case study based on the comparison of precision and recall measures of existing tools for the management of system artefacts and the semantic-based matchmaking process must be defined in advance.

A reuse discovery process can be seen as a search system in which a query (a source system artefact or resource) and a set of resources (a target system artefact) is given. It is necessary to find out which is the best set of results ("mappings") for the source system artefact in the target set. To do so, the following steps must be carried out:

1. Design a domain-based vocabulary,  $O$ , to represent the concepts and relationships that will be used to create the system artefacts. It will usually contain a glossary and the set of metamodels that are used to describe the system artefacts.
2. Select the tools that are currently managing the system artefacts and establish the methods available to discover similar system artefacts. Usually, these tools have not been designed to perform search processes and they just include keyword-based or metadata-based search functionalities that are in general quite simple and restricted for the purpose of reuse system artefacts.
3. Select and design a set of source system artefacts accomplishing with the selected vocabulary and metamodels,  $SR = \{R^1, R^2, \dots, R^k, \dots, R^m\}$ , where  $\#R^k$  represents the number of system artefacts in the specification  $R^k$ . This set of system artefacts will be used as "queries".
4. Select and design a set of target system artefacts accomplishing with the selected vocabulary and metamodels,  $TR^{R^k} = \{R^1, R^2, \dots, R^k, \dots, R^m\}$ , where  $\#TR^{R^k}$  represents the number of system artefacts that are expected to be reused for the system artefact  $R^k$ . This set of system artefacts will be used as repository.
5. Run the reuse discovery process implemented on top of the selected tools (depending on the search capabilities) and the new implementation of the semantic-based matchmaking to discover the matches between the system artefacts in  $TR^{R^k}$  and  $SR$ . For every system artefact in  $R^k$  search for the

best set of mappings in  $TR^{R^k}$ . To do so, two matching methods will be used: 1) text-based and 2) concept-based.

6. Extract measures of precision ( $P$ ), recall ( $R$ ) and the  $F1$  score (the harmonic mean of precision and recall) making a comparison of the expected and generated results. Being  $P = \frac{tp}{tp+fp}$ ,  $R = \frac{tp}{tp+fn}$  and  $F1 = \frac{2PR}{P+R}$  where given a set of system artefacts,  $TR^{R^k}$ , and a query, a system artefact in  $R^k$ :  $tp$  (true positive) is “the number of system artefacts in  $TR^{R^k}$  that have been retrieved and represent correct mappings”,  $fp$  (false positive) is “the number of system artefacts in  $TR^{R^k}$  that have been retrieved and represent incorrect mappings”,  $tn$  (true negative) is “the number of system artefacts in  $TR^{R^k}$  that have not been retrieved and represent incorrect mappings” and  $fn$  (false negative) is “the number of system artefacts in  $TR^{R^k}$  that have not been retrieved and represent correct mappings”.
7. Check the robustness of the comparison by performing statistical hypothesis testing finding out which of the available methods in the different tools has the best performance (in terms of the designated measures, usually the  $F1$  score).

#### 5.1.4 The reuse discovery process in AMASS

Within AMASS, the reuse function shall support the semantics-based mapping of standards. That means that the information available within a standard must be a candidate to be represented under the presented paradigm. Queries (considering as a query any kind of artefact) must return the artefacts that match such standard. To do so, it is necessary to represent the standard (terminology, relationships and patterns (if any)) according to the methodology for knowledge management. Once the standard is represented and indexed, the standard is just another asset, a piece of information that can be retrieved through the interface. It is important to emphasize that the alignment of a work product to a standard will again return a confidence value providing a semi-automatic procedure to discover a mapping between any artefact and a standard. Thus, two main approaches can be done:

- The semantic representation of a standard includes links to the required artefacts (typology), required relationships between artefacts, etc. creating an underlying topology of the structure and contents that an artefact must fulfil to be compliant with the requirements of a specific standard. Then, the retrieval process is in charge of mapping an input artefact structure against the predefined structure of the standard.
- A standard can also be used as the context for searching in two ways: 1) input filter of the relationships that must be fulfilled by the artefacts (first filter restrictions and then search) or 2) output filter to remove those system artefacts that are not fulfilling the requirements expressed in the standards (first search just similar system artefacts, then filter out those that are not fulfilling the standard). It seems that the first approach is more suitable if a standard is driving the reuse discovery process.

With regards to reuse discovery and, more specifically, use of standards as a mean to discover reusable assets, these will represent the required structure (concepts and relationships) that an artefact must fulfil to be a candidate for reuse against such standard. For instance, assuming a standard is represented and indexed within the system, two different artefacts generated for different domains may match the standard description under a certain value confidence. This does not mean automatic reuse but discovery of potential matches. It must serve as an assistant, a recommendation engine, for the end-users.

#### 5.1.5 Definition of an interface for reuse discovery (\*)

As a result of previous studies, an evolution of the preliminary OSLC KM (Knowledge Management) specification has been proposed to define a common shape for any type of system artefact (considered a knowledge asset) that must be represented, stored, shared or exchanged between tools in a development process. On the other hand, the OSLC initiative is making a strong commitment to apply the principles of Linked



Data, RDF (Resource Description Framework) and REST (Representational State Transfer) to boost interoperability.

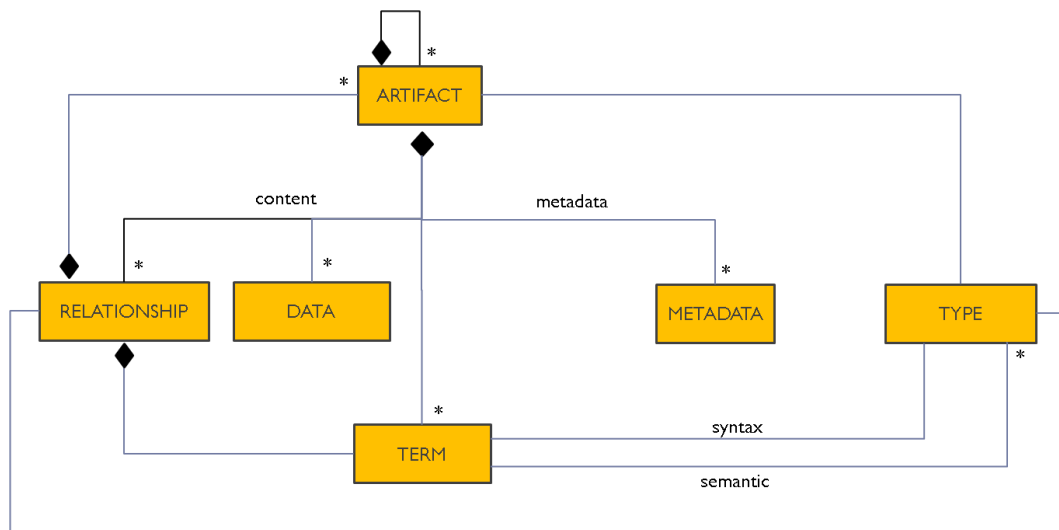
Specifications or, more precisely, data shapes, have already been defined to model metadata and content of requirements, assets, test cases, changes and estimation and measurement metrics. In the same way, the OMG group is working on the OSLC-MBSE specification to promote system models to Linked Data. However, there are still some artefacts for which there is no shape, such as an element of a vocabulary, a requirement pattern or a Dynamic System Model. Due to this fact a common strategy for knowledge management is hard to draw. Moreover, some cross-cutting services such as indexing and retrieval processes are delegated in third-party tools, preventing the implementation of one of the cornerstones of knowledge management for software reuse: selection. Therefore, we present a data shape for any artefact generated during the development lifecycle.

1. SRL: the language for representing the metadata and content of any artefact.
2. Knowledge MANAGER: the basic tool that provides the required services for the software knowledge repository.
3. RDF data shape and OSLC interface: the SRL language is offered through an input/output OSLC interface, satisfying the need of reusing standards in a web environment.

As it has been previously outlined, and in order to combine RDF and SRL, it is necessary to provide an RDFS/OWL ontology, i.e. an RDF vocabulary, that defines the entities and relationships in the SRL representation model to make this specification publicly available and to enable the expression of any piece of knowledge using SRL. On the other hand, and since a huge amount of data, services and endpoints based on RDF and the Linked Data principles are already publicly available, a mapping between any RDF vocabulary and SRL is completely necessary to support backward compatibility and to be able to import any piece of RDF data into RSHP. In this case, taking into account the guidelines and definitions of the OSLC Core specification, the data shape for knowledge management will conform the next basic OSLC definitions:

1. *“An OSLC Domain is one ALM (Application Lifecycle Management) or PLM (Product Lifecycle Management) topic area”*. Each domain defines a specification. In this case, a new domain has been defined: Knowledge Management (KM).
2. *“An OSLC Specification is comprised of a fixed set of OSLC Defined Resources”*. The key concepts of the SRL metamodel are the Artefact and Relationships classes.

An artefact is a container of relationships (RHSP) that can have metaproperties (authoring, versioning, visualization features and, in general, provenance information) and attribute-value expressions (AOV). If an artefact only represents the apparition of a term it will contain a reference to a term (element of a controlled vocabulary or taxonomy). This term can have a grammatical category (Type) such as name, pronoun, adverb or verb to cite just a few. In the same manner, a semantic category (Type) represented by a term can be assigned to a term, for instance the semantics “negative”. Thus, different terms can have different semantics. Finally, a relationship establishes a link between  $n$  artefacts and semantics can be also attached to the link, e.g. “part-of”.



**Figure 11.** UML Class Diagram of the OSLC Knowledge Management Resource Shape.

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

In this case and following the previous design, a shape for every class has been defined. The next table presents the resource shape links to the official definition (prefix:name, e.g. `oslc_km:Artefact`) and a brief description of the resource.

**Table 5.** OSLC Resource Shapes for OSLC Defined Resources within the KM Domain

Class in Figure 11	OSLC Resource Shape Item	Description
Artefact	<code>oslc_km:Artefact</code>	A container of relationships between concepts and metaproperties to semantically describe any piece of information. It is the basis for the creation of an underlying semantic network.
Relationship	<code>oslc_km:Relationship</code>	A relationship represents a link between any set of resources. It is possible to add semantics and it can contain any number of elements representing binary, ternary or even n-ary relationships.
Data	<code>oslc_km:Data</code>	An attribute-value expression that represents a property of the artefact under description.
MetaData	<code>oslc_km:MetaData</code>	A tag-value attribute representing typical metadata properties. Dublin Core is used here to represent such information. Both can be any type of resource or, more specifically, concepts.
Term	<code>oslc_km:Concept</code>	This concept follows the semantics and shape of a <i>skos:Concept</i> [50]. More specifically: “the notion of a SKOS concept is useful when describing the conceptual or intellectual structure of a knowledge organization system, and when referring to specific ideas or meanings established within a KOS (Knowledge Organization System)”.
Type	<code>oslc_km:Concept</code>	Everything has a type and a type is a kind of concept coming from a classification. E.g. The types of UML metamodel, such as Class, Use Case, etc.

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

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

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

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

The detailed description of all properties for every defined resource can be found in the public deliverable “Interoperability Specification – V3” of the CRYSTAL project.

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

It shall be able to process any kind of OSLC-based resource or even any piece of RDF by applying the mappings described in the previous deliverable. Once the data is in the OSLC-KM processor, a reasoning process can be launched to infer new RDF triples (if required). Afterwards, data is validated and indexed into the system and software knowledge repository (SKR). On top of this repository, services such as semantic search, naming, traceability, quality checking or visualization may be provided, generating new OSLC KM Resources.

### Delegated operations of the OSLC KM specification

The notion of delegated operation has been already introduced in Deliverable 5.3 [16] and it is recall here. Although the OSLC approach is perfectly valid for exchanging data resources, there is a huge number of interesting functionalities available in the different tools that should be considered as candidates to be reused through interoperability-based services. As a motivating example, if a model has been created in Papyrus, the engineer may want to check the quality of such model with the IBM Rhapsody capabilities, so the next questions arises: how can we expose functionalities of existing tools in terms of OSLC concepts (enabling operations)?

This is a topic that has been widely studied in the field of web services where standards such as WSDL (Web Services Description Language) and SOAP (Simple Object Access Protocol) were defined to establish a standard way to invoke functionalities via internet protocols (operation-oriented service). The success of web services comes with a lot of APIs already available so an approach for reuse may consider the possibility of invoking existing services but following the principles of an interoperable environment: a common and shared model and a communication protocol.

In the first case, WSDL-based services define a metamodel of the data to be exchanged via an XML-Schema. Each service defines their own XML-Schema so, in some cases, the concept of interoperability is hard to reach because mappings between the data generated from the service and the model of the consumer must be aligned. To ease this operation, also known as “grounding”, semantic web services emerged to provide a



common model, an ontology that would be translated into the specific providers. However, the reality showed that the time and effort to transform an abstract model (the ontology) to a specific model (XML-Schema) was not efficient. Furthermore, it has an implicit implication since this kind of transformation occurs under different levels of knowledge representation (logics vs object models). Secondly, the SOAP protocol is basically an HTTP Post request with attachments. In general, this is a standardized protocol that works perfectly. However, the effort to create and consume requests is greater than the mere invocation of an URL via HTTP.

On the other hand, it is also possible to find the notion of “Delegated User Interface” (DUI) in the OSLC specifications. The main objective of the DUI is to provide a better usability experience for third-party consumers of OSLC services. When you must select or create a resource you can use the native interface of the OSLC provider instead of creating a new one. The unique requirement is that the OSLC provider must have an HTML-based interface. The way of accessing a DUI is natively specified in the OSLC Service description.

Considering the need of keeping backwards compatibility with existing WSDL-SOAP services (even others under protocols such as JSON-RPC) and the notion of DUI in OSLC, we define here the concept of “Delegated Operation”<sup>8</sup> as a function that is exposed by an OSLC service in terms of OSLC resources. It represents a kind of gateway between existing functionality and an OSLC-based environment (Linked Data+REST). In this way, the proposed approach is a hybrid method to expose resource and operation-oriented services.

In the context of the OSLC KM specification, the delegated operations of an OSLC KM provider shall accomplish the following requirements:

- The procedure/function shall be already available in a service provider.
- Generalization of the “*Delegated User Interface*” OSLC concept.
- A delegated operation shall describe its interface like a WSDL service.
  - Host/Port
  - Input parameters
  - Output
- An OSLC KM provider shall implement a system knowledge repository (SKR) comprising a “System Knowledge Base” (SKB) and a “System Assets Store” (SAS).
- A delegated operation shall receive as input an OSLC KM artifact.
- A delegated operation shall generate as output an OSLC KM artifact or a value with a simple data type.
- A delegated operation shall serialize data following the normative OSLC formats (RDF/XML and RDF/JSON) and JSON.

Furthermore, it is necessary to define how to wrap existing operations as an OSLC service. If the operation is available through a WSDL/SOAP interface, which means a host/port, input parameters and output, the interoperable service shall be defined in terms of resources instead of operations. To do so, it is possible to find approaches to create proxies between SOAP and REST services. For instance, this is the case of exposing SOAP services as resources in the IBM Bluemix cloud platform<sup>9</sup>. However, here the focus is to redefine the operation in terms of resources, being the operation by itself a kind of service provider that takes some input parameters and returns some results. That is why it is necessary to make a mapping between the different data types:

---

<sup>8</sup> A formal definition of the delegated operation can be taken from the WSDL W3C Recommendation.

<sup>9</sup>

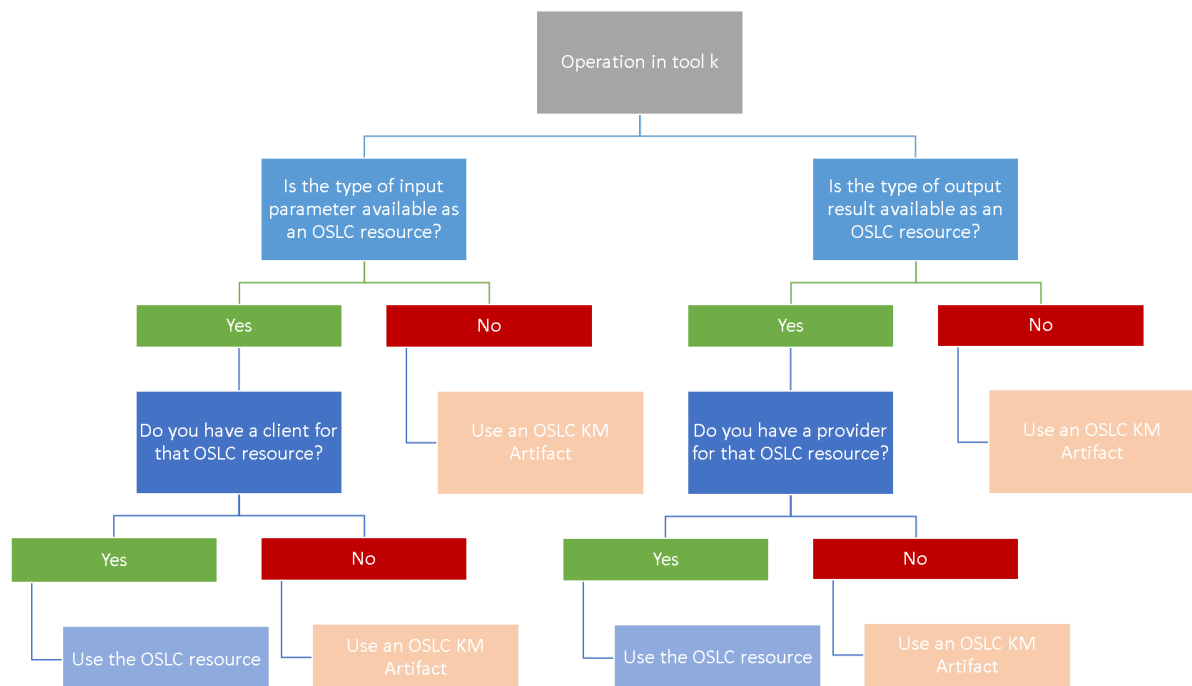
[https://www.ibm.com/support/knowledgecenter/en/SSFS6T/com.ibm.apic.apionprem.doc/tutorial\\_apionprem\\_expose\\_SOAP.html](https://www.ibm.com/support/knowledgecenter/en/SSFS6T/com.ibm.apic.apionprem.doc/tutorial_apionprem_expose_SOAP.html)



**Table 6.** Mapping of WSDL/SOAP operations to OSLC concepts.

WSDL/SOAP primitive	OSLC concept	OSLC KM
Operation $p$ in tool $k$ returning an OSLC resource	Service provider $p$ within the OSLC interface of tool $k$	Service provider $p$ within the OSLC KM interface of tool $k$
Output value $o$ of type $t$	OSLC resource of type $t$	OSLC KM Artifact
Collection of output values $oc$ of type $t$	Collection of OSLC resources of type $t$	Collection of OSLC KM Artifact
Input parameter $p_k$ of type $t$ in operation $p$	Filter $p_k$ of type $t$ in operation $p$	Filtered value or collection of OSLC KM Artifact

In general, the decision tree of what type of resources should be used is presented in Figure 12. The main objective is to reuse both existing operations and OSLC resources. However, if there is no OSLC resource shaped defined for a type of artifact, the OSLC KM resource shaped can be used instead of enabling a better reuse, since there is no need of defining new shapes (clients and providers), but just the mapping rules between the source type of artifact and the OSLC KM Artifact.


**Figure 12.** Decision tree to expose existing operations in a REST-oriented fashion.

According to these logical mappings, let's apply the mappings to an operation ("check") available in a tool, IBM Rhapsody, which receives as input parameter a model and the type of checking, for instance "deep" and returns a set of quality metrics for such a model. This definition would be a function defined as follows:

*check: model x configuration  $\rightarrow Q$  where  $Q$  is a set of key/values.*

In terms of OSLC KM, the function would be defined as follows:

- "check" is a service provider that takes as an input an OSLC KM Artifact representing the model and an OSLC KM Artifact representing the configuration or input parameters as metadata. The output will



be then an OSLC KM Artifact representing the result set Q as a set of data (key/value) within the output Artifact.

In this specific case, it would be also possible to use OSLC KPI (Key Performance Indicators) as output results, but for the shake of a better understanding we define the operation in terms of the OSLC KM resource shape.

Once the notion of delegated operation has been outlined, a set of reusable operations are defined for the OSLC KM providers. More specifically, the next table presents the expected operations available in an OSLC KM provider with support or management of system knowledge bases (SKB) and system assets store (SAS). These operations are expected to ease the reuse of existing artefacts and their metamodels, by providing an implementation for the classical reuse principles of abstraction and selection.

**Table 7.** Delegated operations for an OSLC KM provider (SKB).

Delegated operation in SKB	
<b>Base URI/prefix</b>	http://www.reusecompany.com/oslc/km/operations
<b>Reuse</b>	<base_uri>/skr Query params: operation = {diff, merge, copy} Body params: content = {srl}
<b>Index</b>	<base_uri>/skb/index Query params: type={text   srl   table} Body params: content={content}
<b>Trace</b>	<base_uri>/skb/{id}/trace Query params: type={trace type} to = {id}
<b>Visualize</b>	<base_uri>/skb/{id}/visualize
<b>Normalize*</b>	<base_uri>/skb/normalize Query params: type={text   srl} Body params: content={content}

**Table 8.** Delegated operations for an OSLC KM provider (SAS).

Delegated operation in SAS	
<b>Base URI/prefix</b>	http://www.reusecompany.com/oslc/km/operations
<b>Search artifact</b>	<base_uri>/sas/search Query params: query={text} Body params: srl={srl content}
<b>Filter</b>	<base_uri>/sas/filter -Similar to OSLC query capabilities -Similar to Linkedin API to express filters on attributes: {(key=value,)+}

Finally, Figure 13 depicts the elements of the whole functional architecture for an OSLC KM environment. More specifically, the next building blocks and technologies are being used to implement this approach:

- Tool: It is the target tool from which artefacts and operations are expected to be exposed following the OSLC Resource Shape defined for Knowledge Management.
- OSLC KM adapter: It is a wrapper on top of a target tool, *tool<sub>k</sub>*, which must implement the transformation rules from the internal representation format to the OSLC KM resource shape. Currently, there are some available implementations based on .Net, Java and XSLT.

- OSLC KM Provider: It is an OSLC service provider that offers a Linked Data API (Application Programming Interface) to access the artefacts available in *tool<sub>k</sub>*. Currently, there are two implementations for .Net and Java.
- OSLC KM Client & Provider: It is an OSLC client and Provider for OSLC KM resources. There are again two available implementations in .Net and Java.
- CAKE (*Computer-Aided Knowledge Environment*): It is an API on top of the Knowledge Manager (KM) tool and a repository that offers natural language processing techniques and ontology management capabilities to promote any kind of resource to a semantic-based representation creating an underlying knowledge graph. The CAKE v18 has been used to implement this functional block.
- KM: It is the acronym of the Knowledge Manager<sup>10</sup> v18, a commercial tool developed by The Reuse Company, that offers capabilities to design ontologies and a semantic-based retrieval engine based on graph-matching techniques.
- Common services: Once any piece of data and information is stored in the repository as a graph, it is possible to reuse some of the operations available in the KM tool such as naming, traceability recovery, quality checking or semantic retrieval.

One relevant implication of this architecture is that the reuse of a new type of system artefact or operation only requires the implementation of an OSLC KM adapter and all common services, including reuse operations, will be already available.

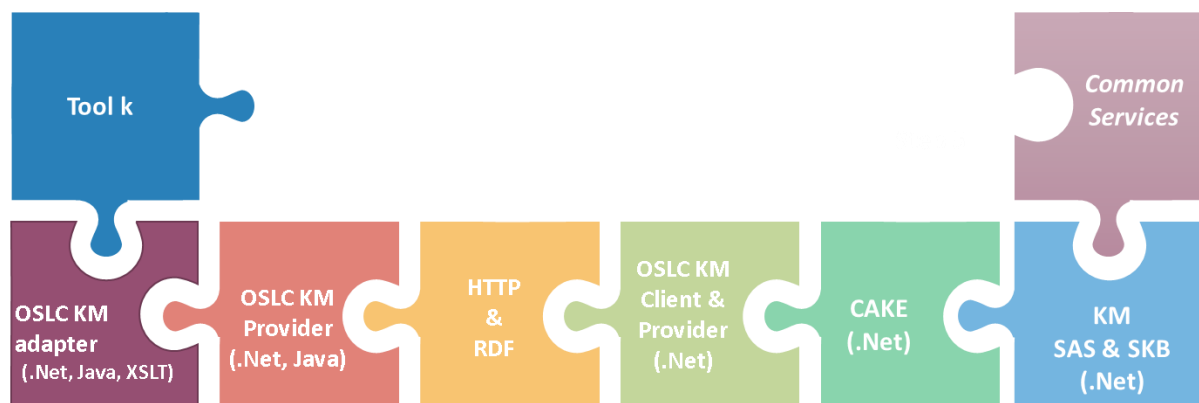


Figure 13. Building blocks of the functional architecture and technology for an OSLC KM environment.

## 5.2 Reuse assistance (\*)

The reuse assistance functionality concerns intra and cross-domain reuse of assurance and certification assets. 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.

The concrete scenarios of reuse include (see Figure 14):

- **Cross-systems 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. We assume that the reusable assurance assets were compliant with the same standards we target in the new scenario.

<sup>10</sup> <https://www.reusecompany.com/knowledge-manager>

- **Cross-standard reuse (cross-concern or 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). The second standard could correspond to a new standard, a new version of a standard, or a different interpretation of a standard (e.g., by a different certification authority).

In the AMASS Prototype P2, the Reuse Assistant does not cover other reuse scenarios such as COTS or SEooC-like reuse.

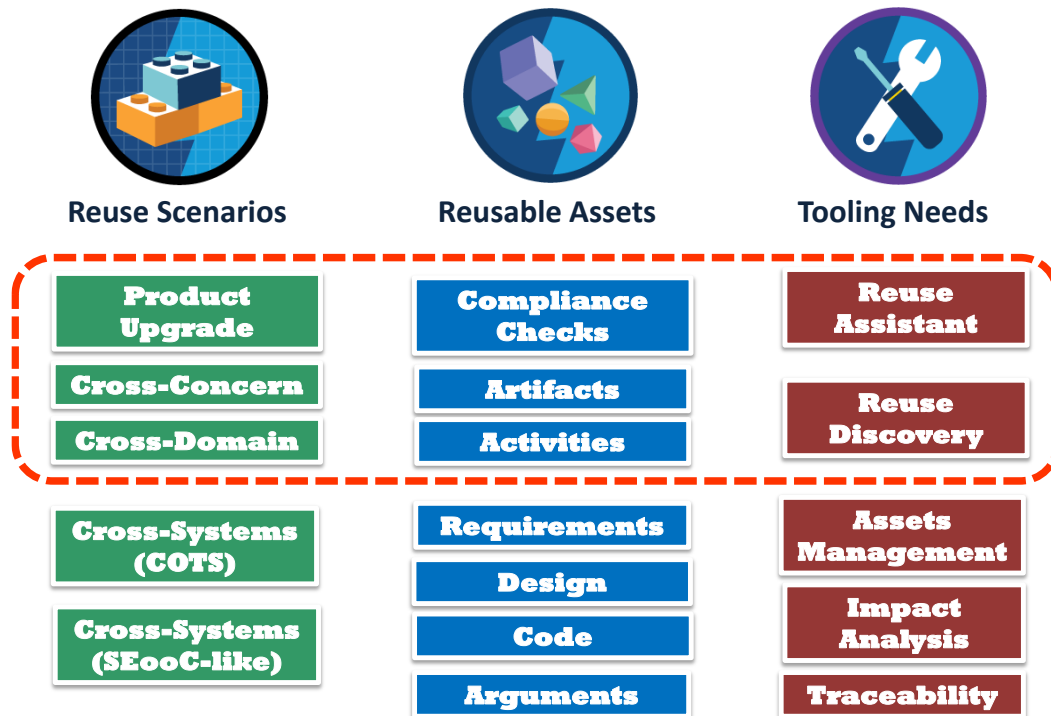


Figure 14. Reuse Assistant: Scope of Reuse in Assurance and Certification

Also, the AMASS Prototype P2 focuses on the reuse of the following 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.

We exclude more product-based assets such as requirements, design artefacts or code.

The reuse assistant tooling will build on top of other functionalities such as reuse discovery, impact analysis, traceability, ontology-based mapping, and assets management.

Figure 15 shows two different approaches for AMASS tooling used in the two scenarios mentioned above.

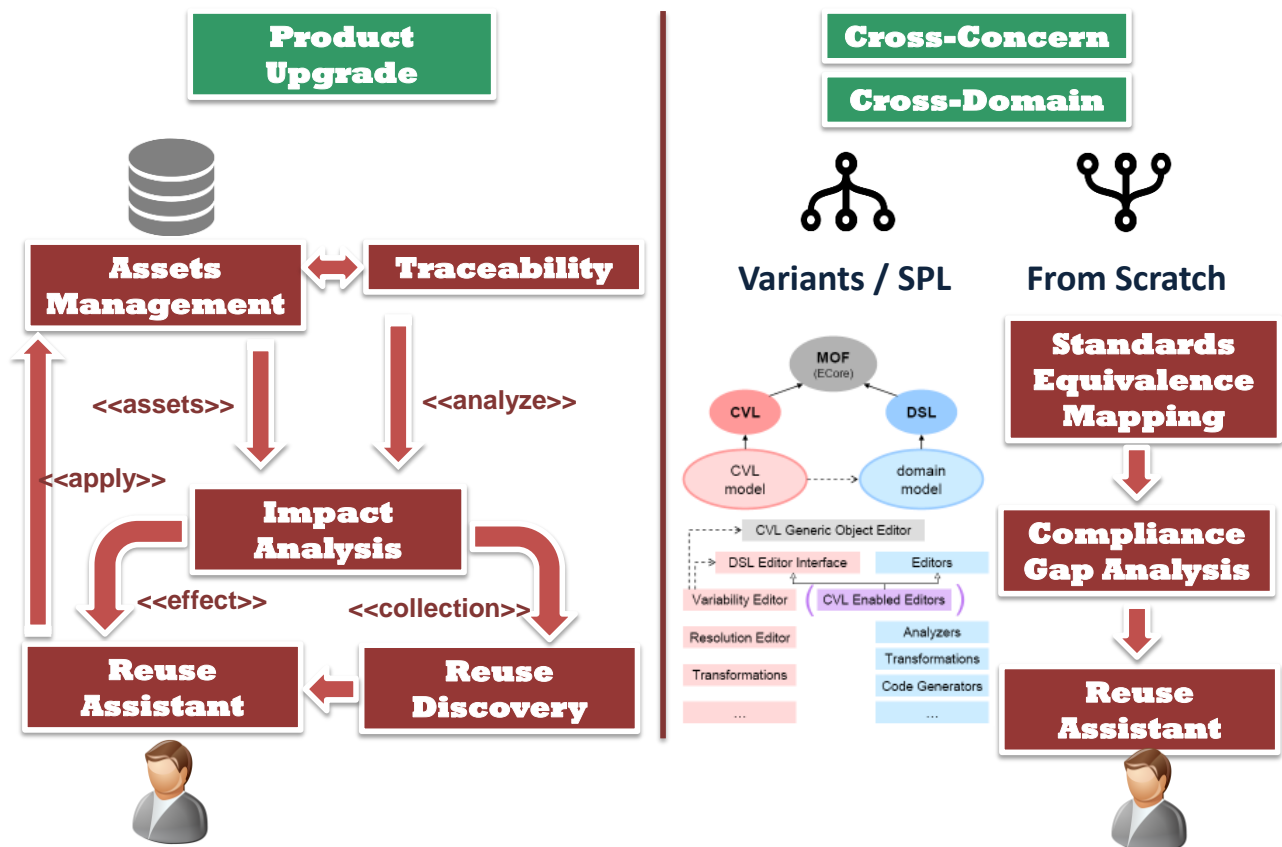


Figure 15. Reuse Assistant: Proposed Reuse Approaches

The decomposition is done at two levels:

#### (a) Layered Decomposition of Tool Services

As a technology-driven decision, we distinguish three tool component layers:

- *Data Management*, including data storage and change management services. This is managed by the CDO technologies in OpenCert.
- *Core Components*, covering the main functionality of OpenCert tool components, and;
- *GUI Client components*, which intend to disaggregate services that can be at some point distributed in separate computing nodes (e.g., when using Web services to access the AMASS platform).

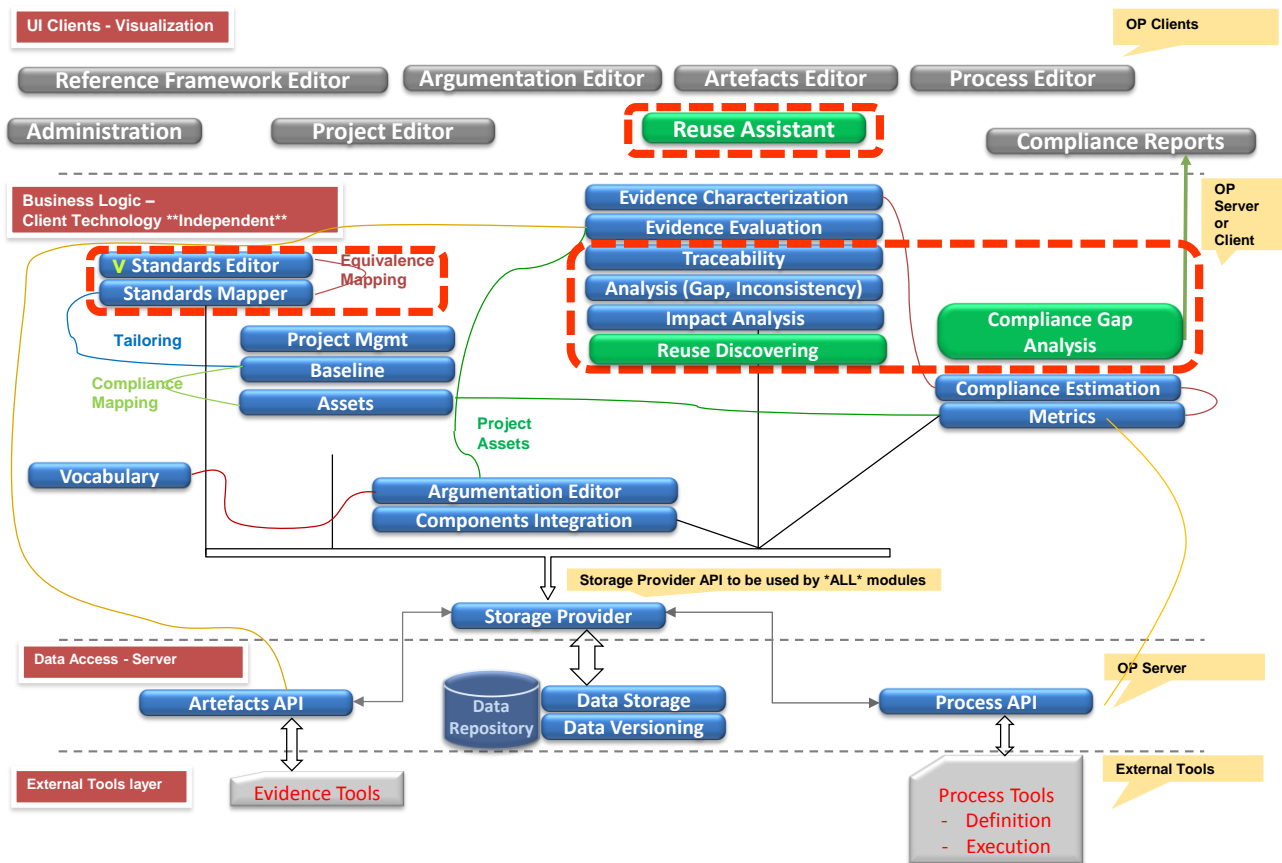
#### (b) Functional Decomposition of Tool Services

The Reuse Assistant is supported on functionalities for:

- Standards mapping, which manages the equivalence mapping between standards models.
- Reuse discovery, which uses impact analysis and traceability management of any AMASS information asset (e.g. argumentation, evidence, or process-related assets).
- Compliance gap analysis, which provides functionalities to understand the compliance gaps of a given baseline.

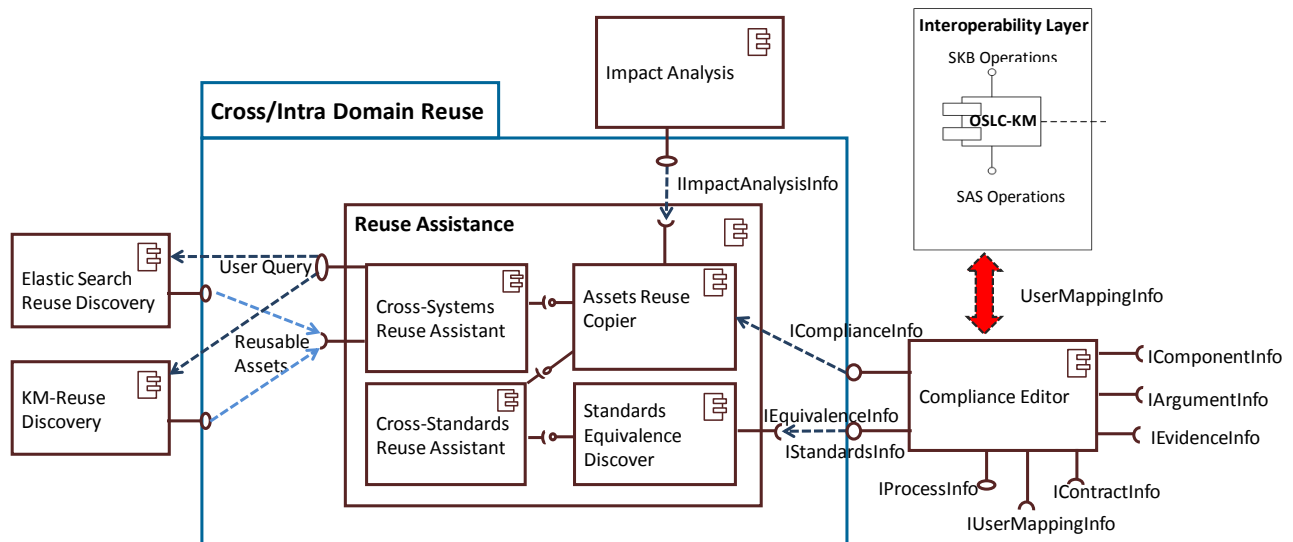
There are two approaches for cross-standard reuse. The reuse assistant supports the “from scratch” approach, explained in section 5.2.2, while the “variants” approach is supported by the families/lines approach explained in section 5.3.

Figure 16 provides more details about the OpenCert architecture that supports the Reuse Assistant functionality. The modules in green support the Reuse Assistant functionalities described in the previous figure.



**Figure 16.** Reuse Assistant: Architecture of new functionalities (in green) in OpenCert tooling

Following deliverable D2.3 [5], which provides a first approach at a high-level of the functional decomposition of the Reuse Assistant module, see Figure 17, we provide a more detailed component architecture as follows.



**Figure 17.** Reuse Assistant: Components decomposition of Reuse Assistant

### 5.2.1 Cross-system reuse scenario

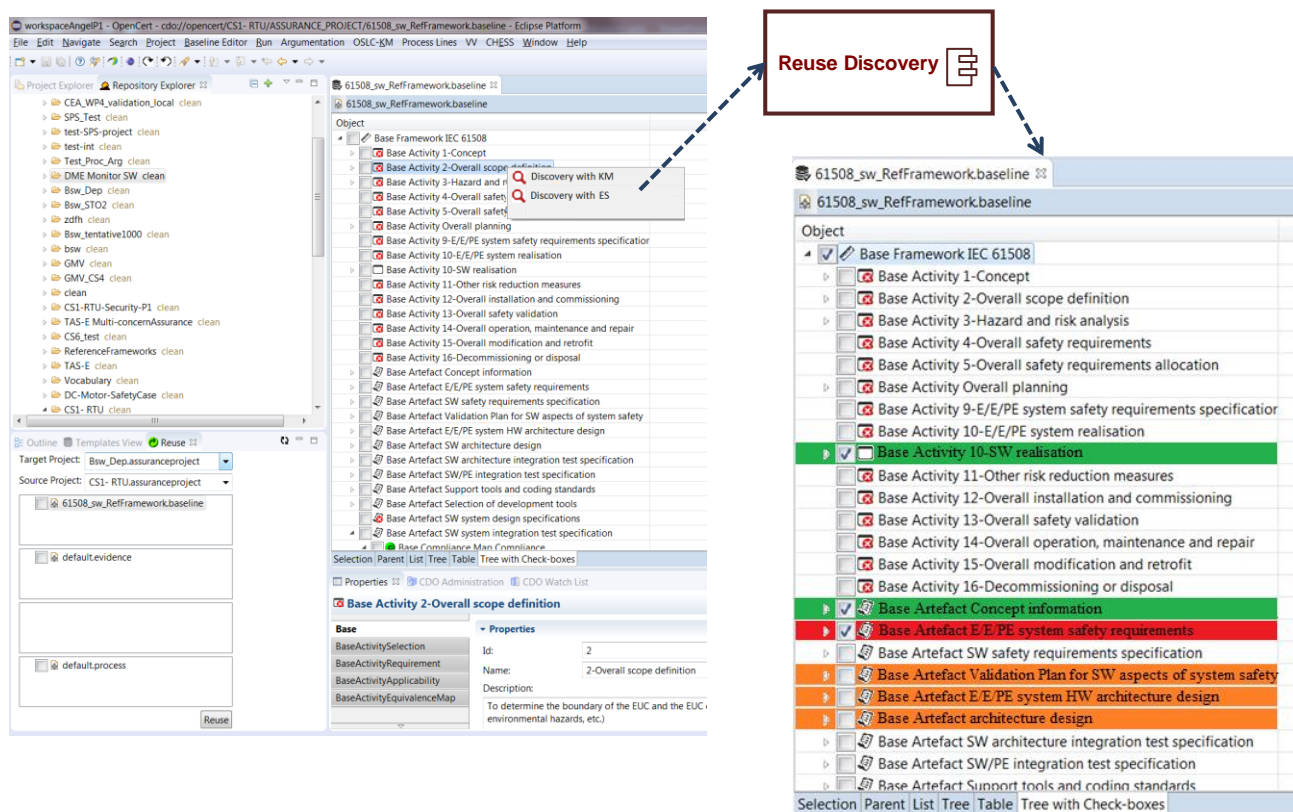
The first functionality relates to the cross-systems reuse (product upgrade) scenario in which both source and target assurance projects must be compliant to the same standard or set of standards. This is managed by the *Cross-Systems Reuse Assistant* component.

To perform cross-system reuse, we use the assets management module. In AMASS, the assets management functionality is provided by CDO as tool for data versioning and storage (see Figure 16).

In this scenario, it is possible to use two reuse discovery approaches. One approach is related to KM module with OSLC technology, as explained in section 5.1, and the second approach is related to elastic search explained in section 5.8. The reuse assistant should use the exposed API, as given by these components, to provide them the query introduced by the user and to show to the user the reusable assets which comply with the specific criteria introduced by a user. The reuse assistant will work with at least one of the approaches (if possible with both).

A possible Reuse assistant interface is shown in the Figure 18, with context menus to let the user select the assets discovery engine and showing the results in a Traffic light way.

- In green. The assets selected by the user are good candidate for reusing.
- In red. The assets selected by the user shouldn't be reused.
- In orange. The asset not selected by the user are good candidate for reusing.



**Figure 18.** Reuse Assistant mock-up: Cross-system reuse scenario

Once the actor selects the assurance assets to be reused, the reuse operation itself can be executed by the Assets Reuse Copier module. The impact analysis functionality (Impact Analysis component, documented in AMASS deliverable D5.2 [15] and in section 5.8) is involved in this scenario in order to understand the consequences of the reuse operation and to identify the set of assurance assets that may be affected when executing the reuse operation.

## 5.2.2 Cross-standard reuse scenario

The second functionality relates to the reuse of assurance assets of one assurance project in another project, when they relate to either different industrial domains, or have different dependability concerns, or different industry-related standards. This is managed by the *Cross-Standards Reuse Assistant* component.





To perform cross-standard reuse, an equivalence map model must be created between the source and the target standard models. This current manual operation could be replaced or assisted by the ontology-based mapping solution, as explained in Chapter 12. This mapping feature is managed by the *Compliance Editor* component and it will be integrated with the ontology base mappings solution. A module for compliance gap analysis allows AMASS users to look at the reuse post-conditions identified in the equivalence map model.

The reuse assistant will help the user in the generation of equivalence maps between standards, showing the data provided by the OSLC-KM module in a Traffic light way, as done in the previous scenario.

- Highlighted in green. The target standard concepts (concepts from Standard B in the tree view on the top right of Figure 19) checked by the user, map (totally or partially) with the selected source standard concept (concept highlighted in grey from Standard A in the left three view on the left of Figure 19) selected by the user.
- Highlighted in red. The target standard concepts (concepts from Standard B in the tree view on the top right side of Figure 19) checked by the user that do not map with the selected source standard concept (concept highlighted in grey from Standard A).
- Highlighted in orange. The target standard concepts not selected by the user (concepts from Standard B in the tree view on the top right side of Figure 19) that do not map (totally or partially) with the source standard concept (concept highlighted in grey from Standard A).

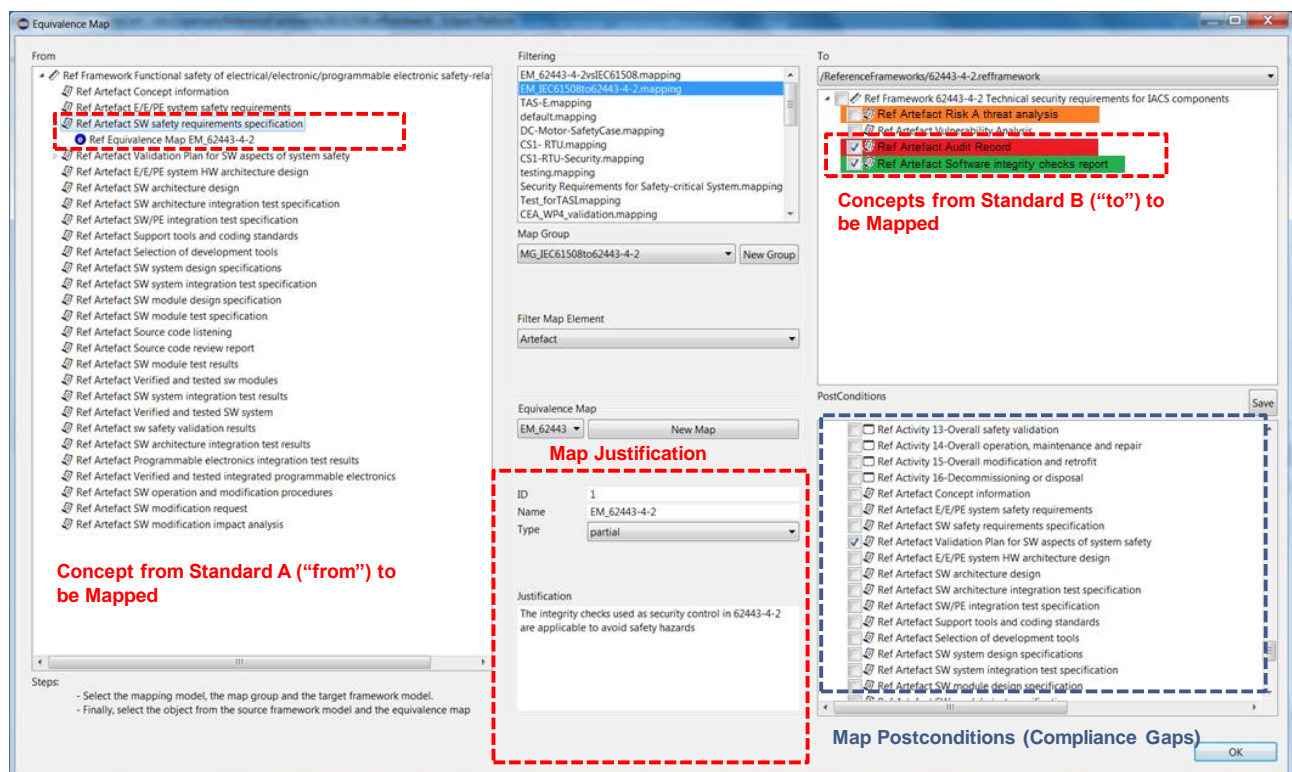


Figure 19. Reuse Assistant mock-up: Supporting equivalence mapping process

The Compliance Editor provides information on the reuse opportunities as result of the analysis of the equivalence and compliance maps relationships between the assurance projects involved in the reuse operation, highlighting in green the good candidate assurance assets to be reused (Figure 20).

Once the user selects the assurance assets to be reused, the reuse operation itself can be executed by the *Assets Reuse Copier*, but the Impact Analysis component is not used for this scenario.





**Reuse Assistant Window**

Target Project: CS1- RTU/ASSURANCE\_PROJECT/CS1- RTU.assuranceproject

Source Project: /CS1-RTU-Security/ASSURANCE\_PROJECT/CS1-RTU-Security.assurancepro

**Equiv. Maps created at standard level in previous stage (see previous Fig.)**

**Assurance Project based on Standard A**

**Assurance Project based on Standard B**

**Map Postconditions (Compliance Gaps)**

**Assurance Assets Created as a Result of the Reuse**

**Reusable Assurance Assets**

Buttons: Reuse, Close

Figure 20. Reuse Assistant: Showing reuse opportunities based on equivalence relations

## 5.3 Management of families/lines

To manage families/lines, it is necessary to have at disposal modelling means for systematizing commonalities and variabilities. These means might be provided as either a specific solution targeting a single type of family (e.g., a process line), or as an orthogonal solution applicable to any type of family. As already and extensively documented in D6.1 [17] as well as in [25], both approaches have been explored in the literature. However, in the context of AMASS, BVR (Base Variability Resolution) turned out to be a promising, feasible, and technically advantageous solution. Thus, in what follows, the essential information regarding BVR is first recalled and then its role in AMASS, for the management of the different families that characterize our problem space, is explained.

### 5.3.1 Base Variability Resolution

BVR (Base Variability Resolution) [38] is a language built on top of CVL (Common Variability Language) [32] to enable variability modelling in the context of the engineering of families of safety-critical systems. BVR is a result of the VARIES project [33]. The specification of the BVR meta-model is given in VARIES D4.2 [42].

BVR enables orthogonal variability management for any model (called Base model) instance of a Meta-Object Facility (MOF)-compliant metamodel. BVR supports the modelling of: feature diagrams, resolution, realization and derivation of specific family members, as well as their analysis. Variability engineers create three kinds of models:

- VSpec models are an evolution of the Feature-Oriented Domain Analysis (FODA) [34]. More specifically, VSpec extends FODA by including additional concepts such as variables, references and multiplicities. Constraints by using the Basic Constraint Language (BCL) can also be added to specify cross-cutting constraints that limit inclusion/exclusion within a subtree based on choices on other subtrees. The grammar of BCL is given in Appendix A.
- Resolution models, which specify the desired inclusion/exclusion choices for the specific configuration/resolution. Note that to confirm whether the resolution corresponds to the VSpec model, a validation process might be executed. The Software Product Line Covering Array (SPLCA) tool is integrated with the BVR bundle for checking constraints and structural consistency of the resolution [24].
- Realization models, which specify the *placements*<sup>11</sup> and *replacements* within the *fragment substitutions*. A Fragment substitution is an operation that, if executed, substitutes a model fragment (placement fragment) with another (replacement fragment). A theoretical exemplification of a fragment substitution is given in Figure 21.

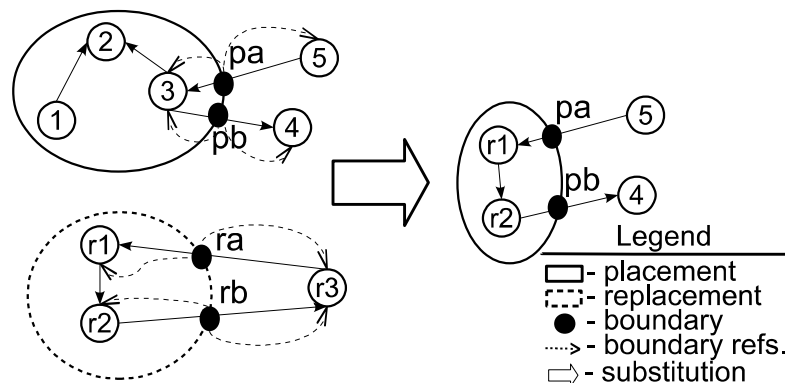


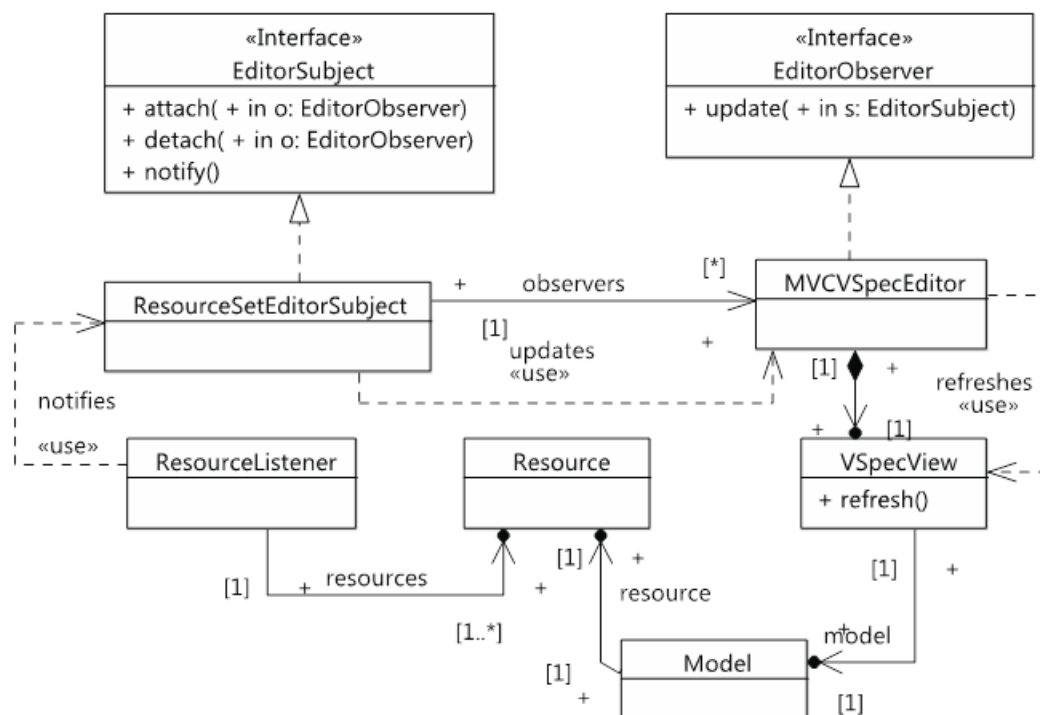
Figure 21. Fragment substitutions exemplification, taken from [40]

<sup>11</sup> A placement fragment is a set of elements forming a conceptual hole in a base model, which may be replaced by a replacement fragment [40].

Some relevant design decisions, presented in [118], pertaining to the implementation of the BVR Tool, are also recalled.

The three different models are supported by three different editors (named VSpec Editor, Resolution Editor, and Realization Editor), each of which reflecting a single underlying variability model. Thus, a variability engineer should be able to resolve, in a resolution editor, a feature just added in a VSpec editor. In addition, the three editors contain views of the model, which should be notified about the model change to keep their content consistent with the model state. Thus, all the editors should share the model resource. Therefore, the editors observe a resource change. Each editor implements the Observer pattern where the resource is a subject. Figure 22 sketches the architecture of the editors in the BVR tool bundle.

In Figure 22, only the VSpec editor (called MVC VSpec editor) is shown. Figure 22 shows that each editor should register itself with a corresponding subject. The subject updates all registered editors when a resource listener notifies about any change to the resource. The editors extend the *EditorPart* class of the Eclipse UI framework. Therefore, Eclipse treats our editors as its own. Each editor contains a view which is a heavy-weight Swing component and renders the underlying variability model. This approach allows us to seamlessly integrate different editors by registering them with a subject that expects a notification from a resource listener. At the same time, the editors can work on their own in Eclipse without any special set up.



**Figure 22.** Editors' architecture

In order to define substitution fragments, an engineer has to select elements in a base model. The base model is typically defined and modified in editors of the target language. To perform selections, the BVR tool has to communicate to the editors of the target language. Therefore, the realization editor is capable of interfacing third-party editors via *IBVREnabledEditor* interface. Thus, editors of the target language have to implement this interface or provide adapters to establish a link between the BVR bundle and targeted editor, see Figure 23. By default, the BVR tool- chain provides integration with any EMF based tree editors and Papyrus UML.

The suggested architecture allows integrating different editors seamlessly. It decouples different components which can be run as stand-alone plug-ins.

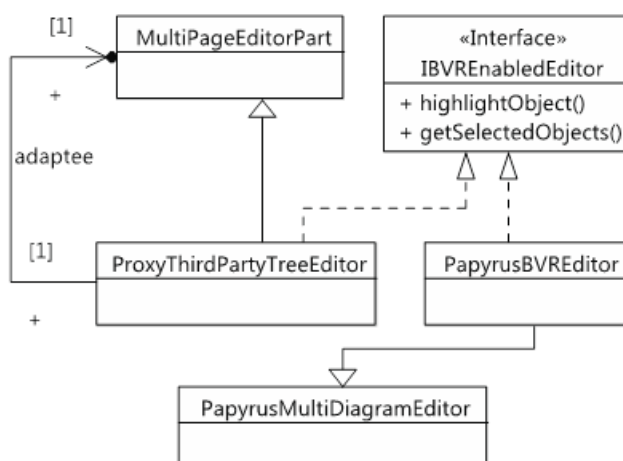


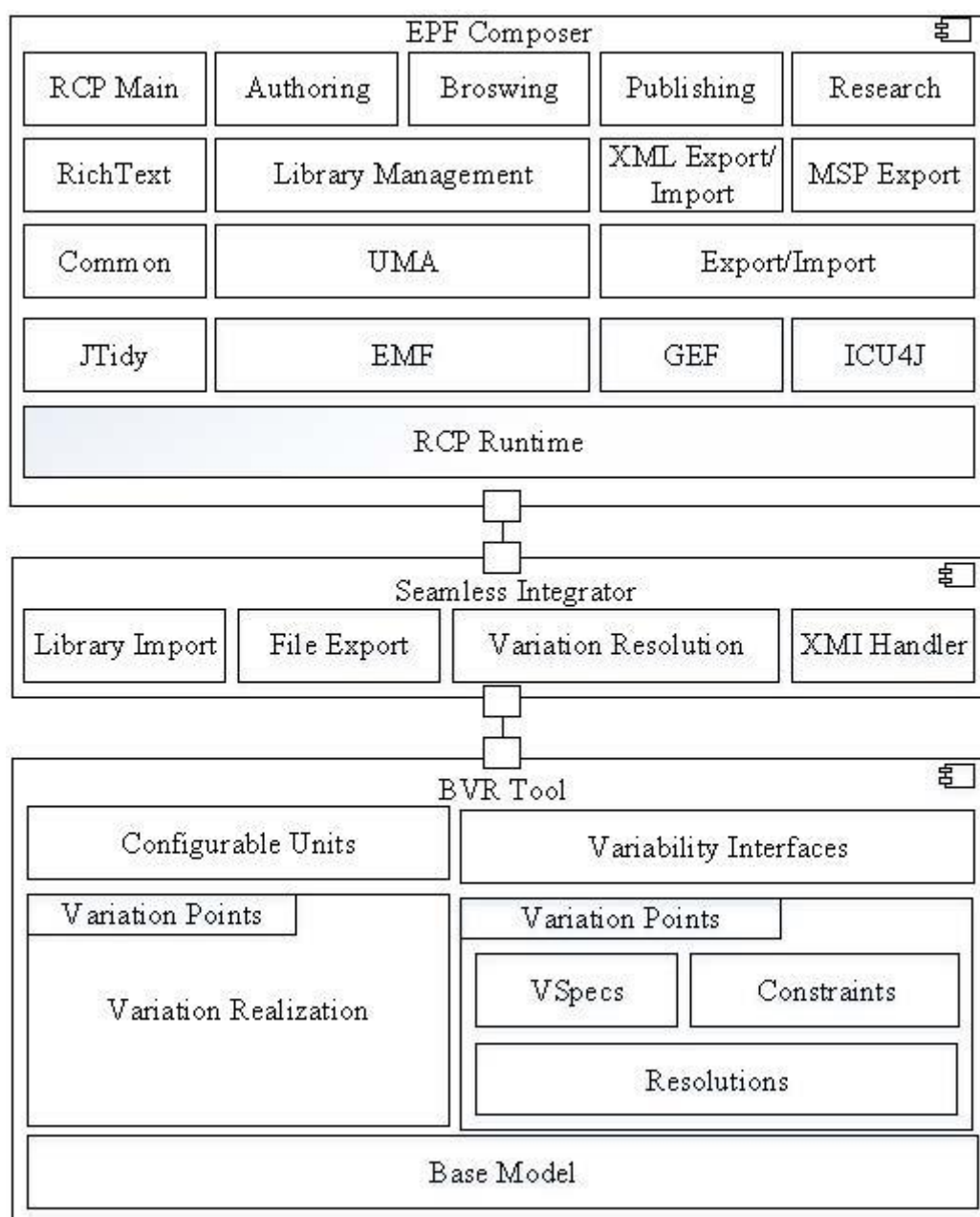
Figure 23. Third party integration

## 5.3.2 Process-related reuse via management of process lines

### 5.3.2.1 Empowering SPEM2.0/UMA to enable management of process variability

In the context of AMASS, SPEM2.0/UMA has been selected to model those processes that represent plans. Models that represent execution of plans are, instead, modelled in CCL. Both SPEM2.0/UMA and CCL, however, do not offer any support for managing variability as needed for the AMASS purposes.

To be able to automatically reuse the UMA-given representations of the process elements, which have been presented in 4.1.1, it is necessary to empower SPEM2.0/UMA by offering means for variability management. To do that, the AMASS solution is BVR. However, seamless integration between the tools that implement UMA (EPF Composer) and BVR (BVR Tool) is not given for granted, and some challenges have to be overcome. The specific challenges and the work conducted to overcome them is extensively documented in D6.5 [20] and partly in [153]. In this deliverable, only the design of the plugin necessary to enable the seamless integration between EPF Composer and BVR Tool is provided. Figure 24 depicts the architecture of the conceived plugin (*Seamless integrator*) and it also recalls the architecture of EPF Composer (detailed in [44]) and BVR Tool (detailed in [42], page 50). As it can be seen from Figure 24, *Seamless integrator* is expected to import all necessary information (Library) from EPF Composer to enable the communication with BVR Tool. It also includes *XMI handler* to handle specific challenges related to XMI files (more details can be found in D6.5). In addition, a functionality enabling the variation resolution is also conceived. Finally, the export back to the EPF Composer is included.



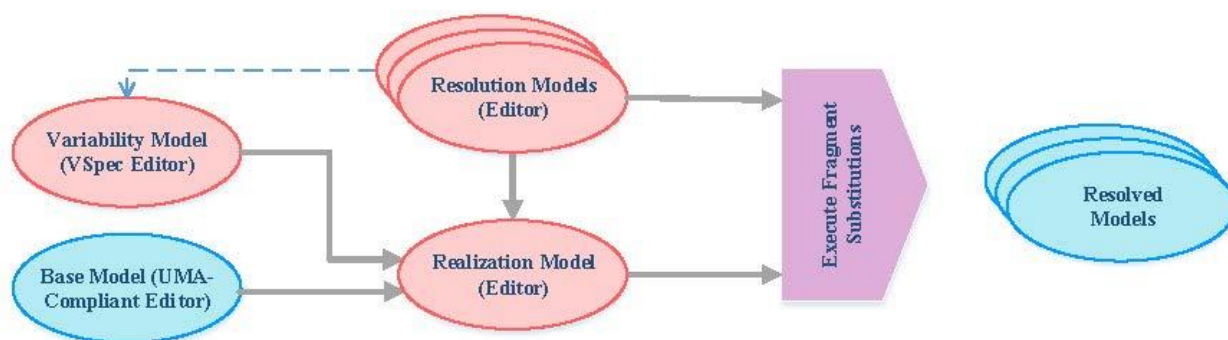
**Figure 24.** Architecture of the seamless integrator plugin enabling process-related variability management

Figure 25 depicts the interplay of the different models needed to manage process variability via integration of UMA and BVR. In particular, process engineers jointly with variability engineers should provide four models:

- a Base Model (an UMA-compliant model), to model the single process to made vary,
- a VSpec Model, to model the feature diagram associated to the process model,
- a Resolution Model, to model the process configuration, and
- a Realization model, to model placements and replacements.

The interplay of these models is then processed/elaborated to produce a resolved model where substitutions have been executed.

Summarizing, BVR provides advanced support for managing families (security-informed safety-oriented process lines, product lines, etc., depending on the specific choice of the base model).



**Figure 25.** Models interplay enabling management of process lines

### 5.3.2.2 Intra-domain variability management: an aerospace SoPL

In this subsection, as a running example, a portion of ECSS-E-ST-40C [37] is considered. Thus, first some essential information is recalled.

ECSS-E-ST-40C is one of the series of ECSS Standards intended to be applied together for the management, engineering and product assurance in space projects and applications. ECSS-E-ST-40C targets software development. More specifically, it covers all aspects of space system software engineering, including requirements definition, design, production, verification and validation, transfer, operations and maintenance. Similarly to other standards, it represents in effect a “standard for making standards”, the idea being that this permits suppliers to use their own standards, provided that they comply with the requirements of ECSS-E-40C or some tailoring of it defined (or at least agreed) by the customer [39]. Tailoring support is thus a strategic ability for enabling customers as well as suppliers to perform valid customizations. Different customizations, performed by the different customers, can be seen as variants within a family of processes. Predefined tailoring rules are provided in a specific annex of the ECSS Standard, Annex R (normative), based on software criticality, which ranges from A (most critical) to D (less critical). Different tailoring choices may also be defined with criteria different from criticality, mainly according to the level of risk which is taken by not performing given engineering activities.

ECSS-E-ST-40C, Section 5 (Software design and implementation engineering process) is constituted of a series of phases (Design of software items, Coding and testing, Integration), each of which containing various activities, which in turn contain various tasks. The phase *Integration* is composed of two activities: *Software integration test plan development* and *Software units and software component integration and testing*. According to Annex R, for instance, the *Software integration test plan* is applicable (Y) for levels A-B, and is also applicable (Y) for level C except SUITP K.9 and K10. Finally, it is not applicable for level D. This limited process portion exemplifies what is typically required in terms of process engineering, i.e., complying with the requirements while tailoring. Thus, enabling valid tailoring is fundamental.

An UMA-compliant base model of this process description is given in Figure 26.





Presentation Name	Ind...	Predec...	Model In...	Type	Plan...	Repeat...	Multiple Occur...	Ongo...	Event-Dri...	Option...
Software Design and Implementation Engineering Process	0			Capability P...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Design of software items	1			Phase	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Detailed the design of each software component	2			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Develop and Document software interfaces	6			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Produce the detail design model	8			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Detail software design method	10			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Detail Design of real-time software	12			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Describe Software Behaviour	18			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Determine design method consistency for real time	20			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Develop and document manual	22			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define and document software unit test	24			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Conduct a review of the detailed design	26			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Coding and Testing of Software Items	28			Phase	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Develop and Document Software Units	29			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Test Software Units	31			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Integration of software	35			Phase	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Develop Software Integration Test Plan	36			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Develop Software Integration Test Plan	37			Task Descri...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Integrate and Test Software Units and Components	38			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Integration and Testing Software Units	39			Task Descri...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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

Figure 26. UMA-based model of the ECSS process fragment

Figure 27, Figure 28, and Figure 29 depict respectively the VSpec model, the Resolution model and the Realization model. More specifically, Figure 27 depicts the feature model associated to the process fragment depicted in Figure 26.

As it can be seen, the process element *software\_design\_and\_implementation\_engineering\_process* has three mandatory sub-features (depicted as rounded rectangles and representing selectable entities, where a choice can take place), one of which is Integration. This sub-feature in turn has 2 mandatory sub-features, which are further developed but at the same time are constrained (note that parallelograms denote constraints). Note that constraints are given by using Basic Constraint Language (BCL), which is an OCL [50] subset (see Appendix A). The understanding of the constraints is out of scope. Specific guidelines will be offered in the user manual. What instead is relevant is understanding that these constraints may represent cross-cutting dependencies (one choice in one sub-branch constrains the choice in a different sub-branch).

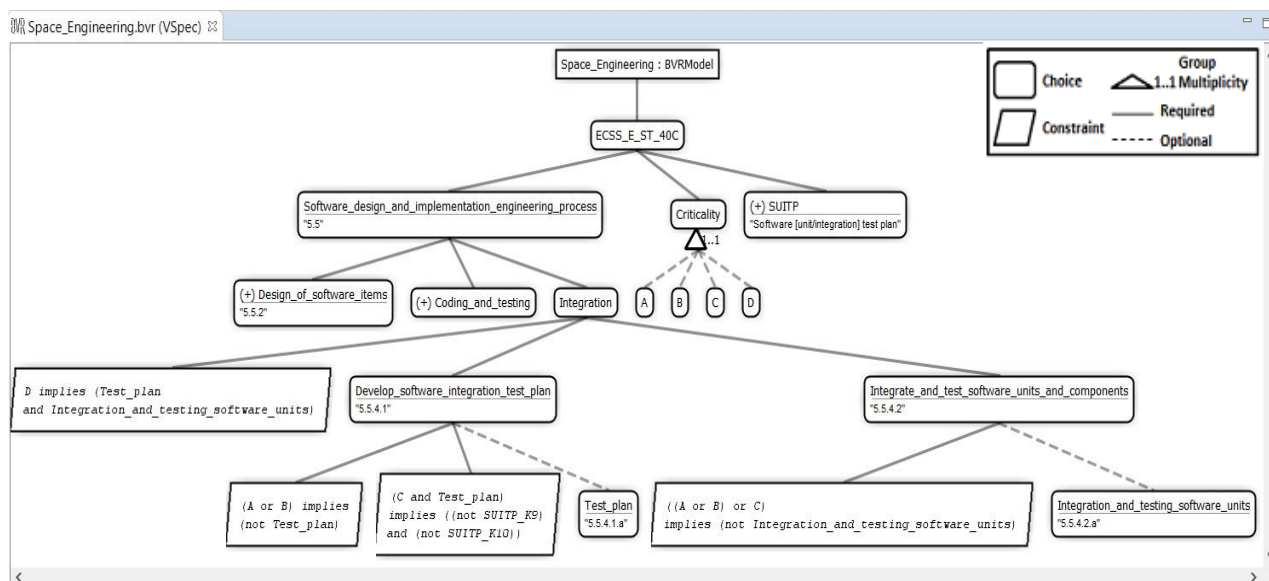


Figure 27. VSpec model regarding a portion of ECSS-E-ST-40C



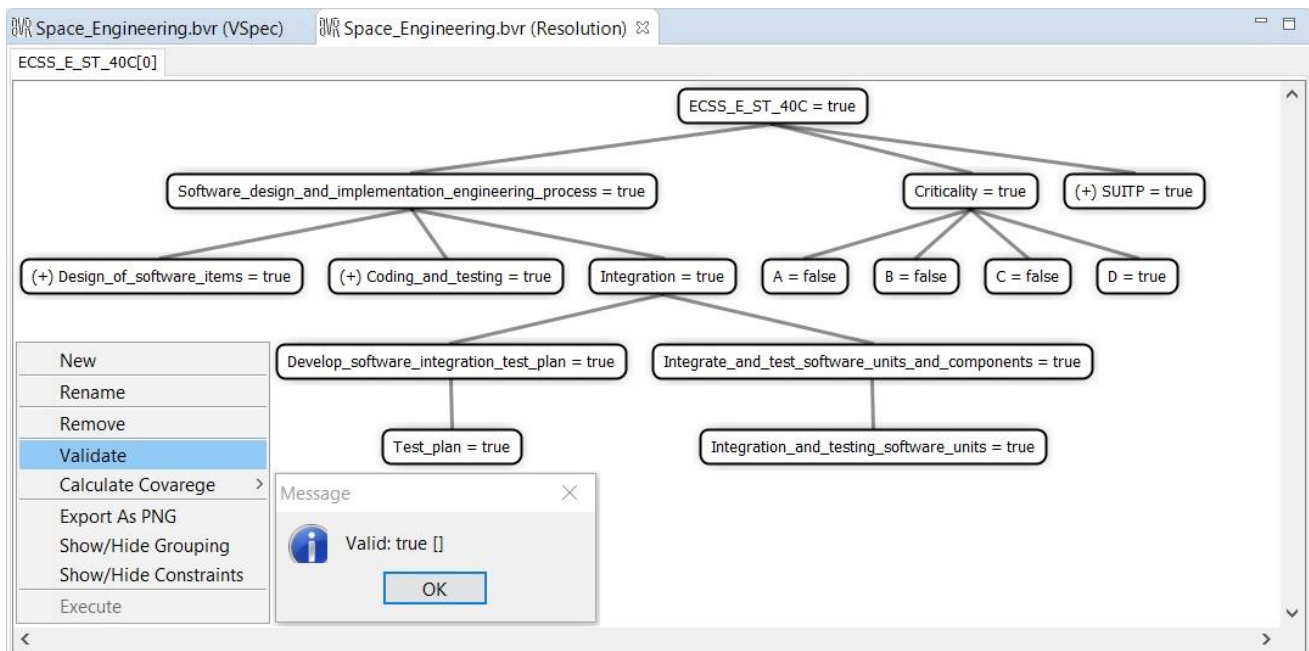


Figure 28. Resolution model

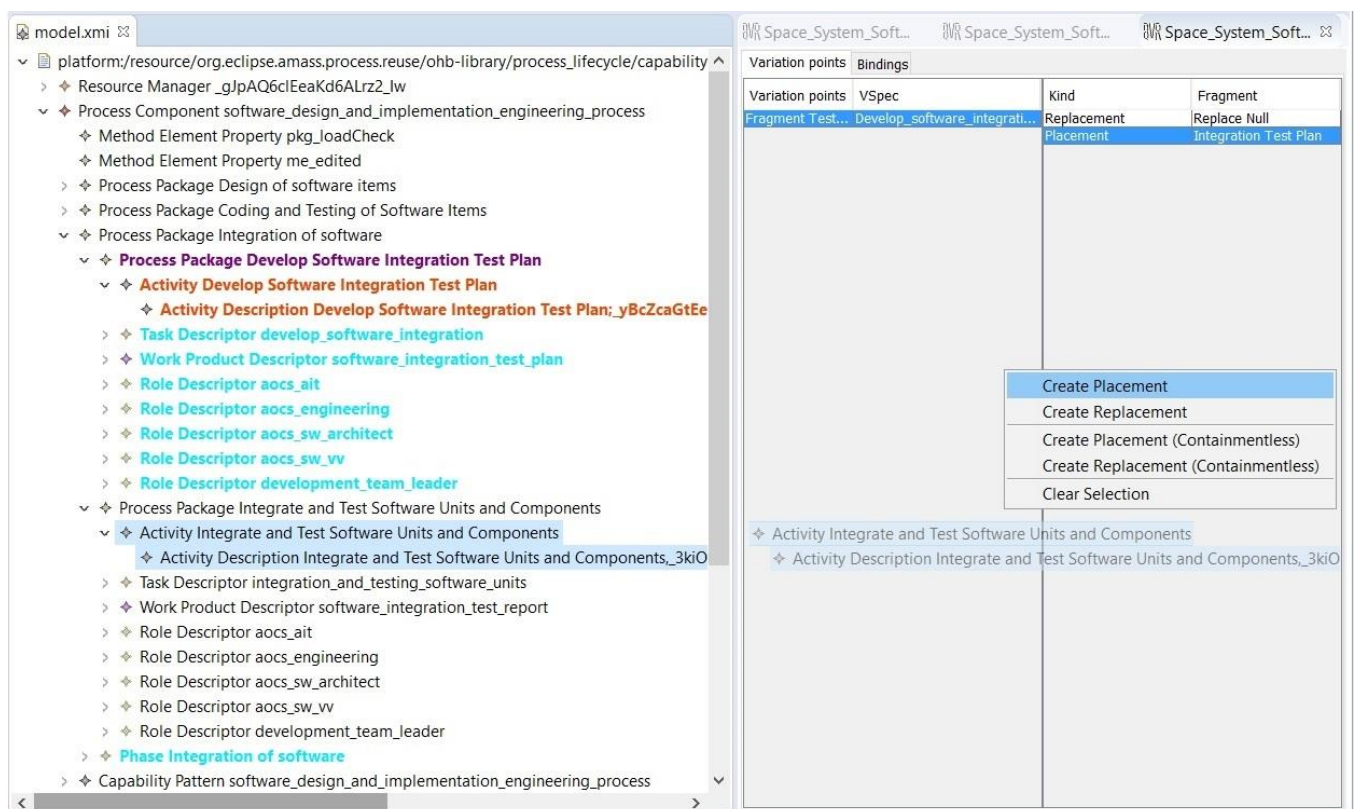


Figure 29. Realization model

Once the replacements (red text) and placements (light-blue text) are defined (see Figure 29) and substitutions are performed, the original UMA-model is replaced by the new tailored model, as depicted in Figure 30.



Presentation Name	In...	Predecessors	Model Info	Type	Planned	Repeatable	Multiple Occurrences	Ongoing	Event-Driven	Optional
Software Design and Implementation Engineering Process	0			Capability Pattern	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Design of software items	1			Phase	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Detailed the design of each software component	2			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Develop and Document software interfaces	6			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Produce the detail design model	8			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Detail software design method	10			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Detail Design of real-time software	12			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Describe Software Behaviour	18			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Determine design method consistency for real time	20			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Develop and document manual	22			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Define and document software unit test	24			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Conduct a review of the detailed design	26			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Coding and Testing of Software Items	28			Phase	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Develop and Document Software Units	29			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Test Software Units	31			Activity	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Integration of software	35			Phase	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 30. Backward propagation of the changes onto the original model

It should be noted that a similar set of models could be obtained to manage ECSS requirements variability. This means that a feature diagram could be associated to represent the requirements variability due to requirements evolution in time, whenever new ECSS-versions are released.

### 5.3.2.3 Cross-domain variability management: towards an automotive-avionics SoPL

Commercial off-the-shelf (COTS) components are very often used in avionic systems. Airborne electronic hardware is composed of simple components like resistors up to highly complex integrated circuits like microcontrollers. The design and development process of these parts does often not follow the recommended guidance DO-254 (Design assurance guidance for airborne electronic hardware) of aviation. Furthermore, the design data of the COTS components is often confidential and usually not available for a review.

The certification process in aviation does not address individual parts or components because these COTS are addressed when the function they belong to is verified. This differs from other domains where individual parts themselves have to be qualified for usage.

However, DO-254 section 11.2 states that the basis for using COTS components in aviation is the use of an Electronic Component Management Process (ECMP), which supports the design and development of airborne electronic hardware. The ECMP of the airborne electronic hardware designer should take care that several aspects of DO-254 are covered and satisfied, independent of the manufacturer of the COTS components. These aspects mainly address the quality, reliability and suitability of the COTS components. Using an ECMP is essential for establishing the pedigree and authenticity of all COTS components that are used.

EASA Document EASA CM – SWCEH -001, section 9 provides additional guidance how to handle DO-254 for certification aspects associated with the use of COTS. Depending on the complexity of the COTS and the assurance level to achieve, different activities have to be performed. For example, a classification of the COTS components has to be done and product and production data have to be collected.

In the context of AMASS UC7, IFX and LAN are concentrating their attention on using automotive semiconductor integrated circuits for aviation. The designer and manufacturer of automotive integrated circuits have several standards to fulfil: AEC Q100 [110], which defines robustness and stress tests to qualify a component; IATF 16949-2016 [111], which is a specification of a quality management system issued by the International Automotive Taskforce; and ISO 9001-2015 [112], which describes requirements for a quality management system. The processes and activities mandated by these standards generate documents and artefacts, which could be reused in the aviation domain to fulfil the DO-254 requirements for using COTS.

Despite the relevance of ISO 26262, Part 5, in the context of WP6, IFX and LAN have decided to limit the investigation to the above-mentioned set of standards.



IFX and LAN have investigated what activities are meaningful, in order to use highly complex microcontrollers designed for automotive applications as COTS components in aviation. The study compared requirements of both domains and identified several actions to provide artefacts for the ECMP aspects of DO-254. Table 9 summarizes the procedure.

**Table 9.** Mapping between avionics and automotive regulations

DO 254 Requirement (11.2.1)	task	result	automotive requirement	comment
1. The component manufacturer can demonstrate a track record for production of high quality components.	document quality performance /KPIs	Quality performance data of the automotive division of Infineon, IATF16949 certificate	ISO 9001-2015 (section 8) and specific regulation listed in IATF 16949 (section 8)	this basically asks for the consistency of the Quality management system of the production
2. Quality control procedures are established at the component manufacturer.	check control plan	list of measures in control plan	IATF 16949; 8.5.1 (control of production)	
3. There is service experience supporting the successful operation of the component.	check field status	delivery since 1 year	Customer requests: statistical bin analysis (SBA), excursion management, complaint management IATF 16949; 10.2.6 (field failure test analysis)	
4. The component has been qualified by the manufacturer or by means of additional testing, which establish the component reliability.	check qualification report	qualification released 12/2015; 40°C to 150°C ambient operating temperature range	AEC, Q100, grade 0 with special focus on appendix 7	AEC Q100 contains detailed stress test description, references for external standards and related success criteria independent of the application, appendix 7 refers to application specific qualification strategy
5. The component manufacturer has control of the component quality level or that this is assured by means of additional component testing.	check final test results and monitoring matrix	final test	IATF 16949; 9.2.2.4 (product audit)	100% final test is standard for automotive product in combination with screening strategies and methods like SBA
6. The components have been selected on the basis of technical suitability of the intended application, such as component temperature range, power or voltage rating, or that additional testing or other means has been used to establish these.	check specification	automotive: chassis safety applications power train applications -40°C - 125°C 3.3V or 5V 200MHz	IATF 16949; 8.2.1.1 (customer communication) 8.4 (Control of externally provided ...products...)	
7. The component performance and reliability are monitored on a continuous basis, with feedback to component manufacturers concerning areas that need improvement.	failure analysis requests (FAR) process monitoring matrix	failure analysis requests (FAR) process wafer and product monitoring	IATF 16949; 8.5.1 (control of production...)	

#### 5.3.2.4 Cross-concern variability management: an automotive SiSoPL (\*)

In this subsection, the usage of EPF-Composer (EPF-C) and the BVR tool to model an automotive Security-informed Safety-oriented Process Line (SiSoPL) [53] is explained. A SiSoPL model related to functional safety (ISO 26262) and cybersecurity (SAE J3061) is defined. Via the SiSoPL model, engineers are able to reuse processes or process elements, even if they vary within defined boundaries. The presented solution uses the integration of EPF-C and BVR-tool, first to model the base model and then to manage variability aspects. A general tool description concerning BVR and EPF-C is already available from MDH (see 5.3.1 Base Variability Resolution).

Figure 31 shows a simplified version of the recommendation tables' template that guide the applicant towards the usage of the ISO 26262 recommended methods. As it can be seen, this template contains two variability points (ASIL and Recommendation Level (ReCL)). It is a company or project specific decision which recommendation level is considered (e.g. for ASIL B the methods with ReCL "+" are executed, see Figure 32). For system design analysis, for instance, within ISO 26262, Part 4, Method-1 and Method-2 are two consecutive entries and correspond to Deductive analysis (e.g., FTA) and Inductive analysis (e.g., FMEA) and their recommendation levels are: (o, +, ++, +++) and (+++, ++, ++, ++).

Methods				
	A	B	C	D
	Method-1			
	Method-2			
	Method-n			

**Figure 31.** Recommendation table from ISO 26262-4

As a first step, during the SiSoPL engineering, the scoping of the family has to take place. Thus, standards and regulations to be taken into consideration are selected, see [52] and [53].

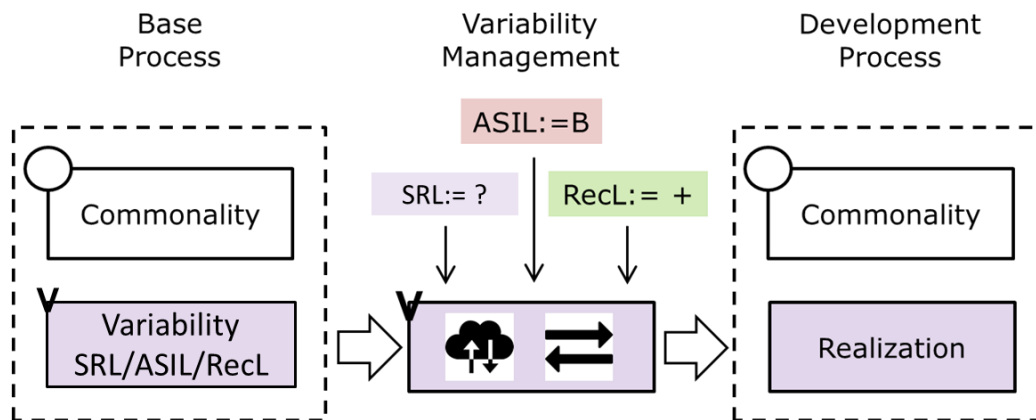
In a second step, a base process model is created. It contains all activities, which can possibly be part of the development process and which may be needed for any ASIL and any Security Risk Level (SRL). This model is directly related to the underlying standards and additional company specific activities. This model is defined in the EPF-Composer.

Step three is tailoring the project specific process. This means that we remove unwanted activities and add new project specific ones. If we consider for instance the concept phase, safety integrity levels (ASIL) and SRL are determined. These variable levels represent the type of variability that will be modelled with the BVR tool. ASIL and SRL vary depending on the item that will be developed. If the development process should be available for various items, it has to deal with variability because different items and even different functions of an item may have different ASILs. This means we need a mechanism to change activities and methods according to different ASIL and SRL. In the presented application we define activities, which consider safety and security. We deal with a cross concern methodology and we have to change some terms, which come from SiSoPL mainly used in a single concern/cross domain perspective.

Activities in the actual cross concern application, which have to be executed in any case, are called safety security co-engineering activities (instead of the single concern “commonality”). The intention is to “maximize” co-engineering activities and deal with variability in a way that makes processes reusable. We have to make sure that co-engineering guarantees interaction between safety and security related activities. They have to use co-engineering capable methods, which can deal with both areas. In addition, we have safety and security specific activities, which depend on ASIL and SeCL.

To demonstrate the approach, we define and model a short example in EPF-Composer concerning the concept phase of the safety and security lifecycle. The integrated process considers safety and security aspects. An

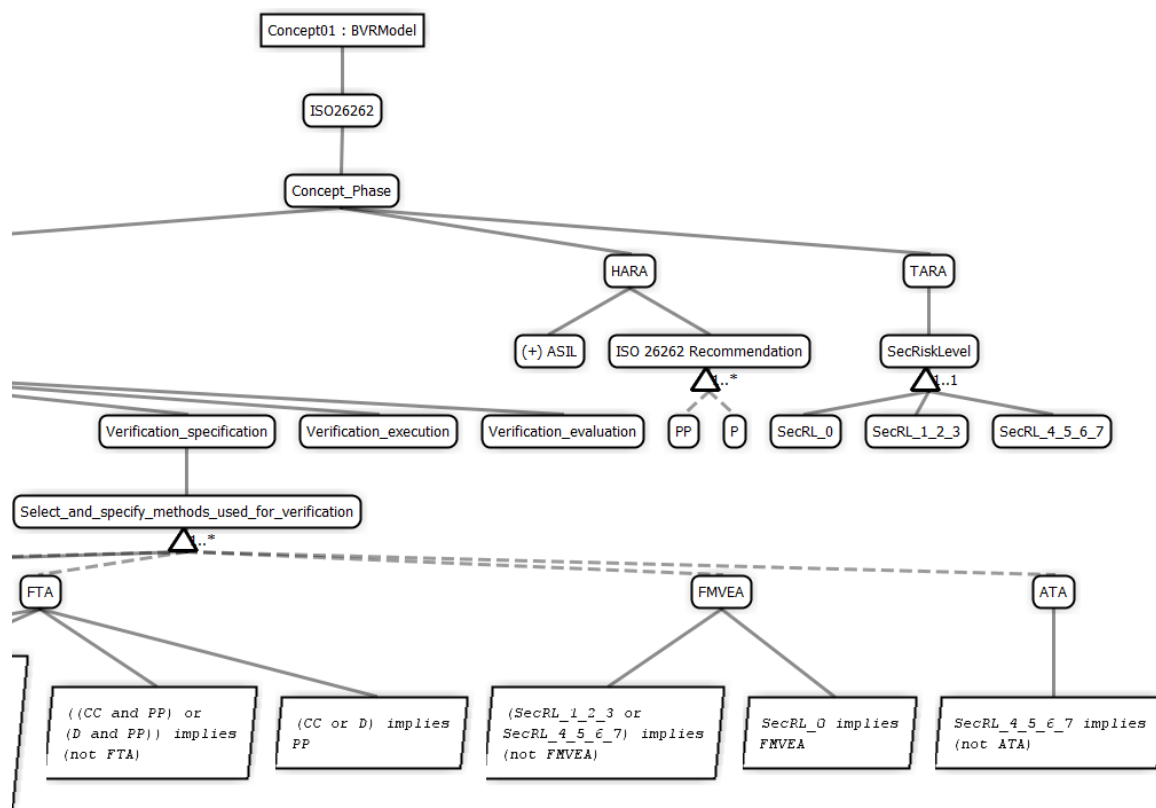
activity “risk analysis”, which contains safety and security analysis, is defined. We consider risk analysis in this case as a co-engineering activity, because it has always to be performed by safety and security considerations. The risk analysis activity includes specific methods, in this case HARA for safety- and TARA for security-related analysis. The output of the risk analysis may lead to different safety and security risk levels. This fact leads to process variability, because the outputs of the analysis are safety- and security-goals and corresponding requirements. The identified safety and security requirements have to be allocated to the available customer requirements. HARA leads to a safety integrity level, whereas TARA, which identifies the highest risk potential threats, leads to a security risk level (SRL). For this reason, the process model needs to support that kind of variability. This means that activities demanded by specified criticality levels (ASIL/SRL) have to be changed specifically.



**Figure 32.** Variability management, ASIL B and Recommendation level “+” are determined, SecL is not defined

Modelling of the variability in the example is done with the BVR-tool in the Eclipse environment. Based on the EPF-C model, which contains all available activities without any criticality level, we have to define a VSpec model that covers the variability concerns. This VSpec model contains only varying activities, which are related to ASIL, SRL, and RecL. This means that only alternatives (XOR) and optionalities (0/1) are part of the model. This is shown in the Resolution diagram Figure 35 which is nearly identical to the VSpec diagram (see Figure 33). A detailed description concerning VSpec- Resolution- and Realization diagram can be found in section 5.3.2.2.



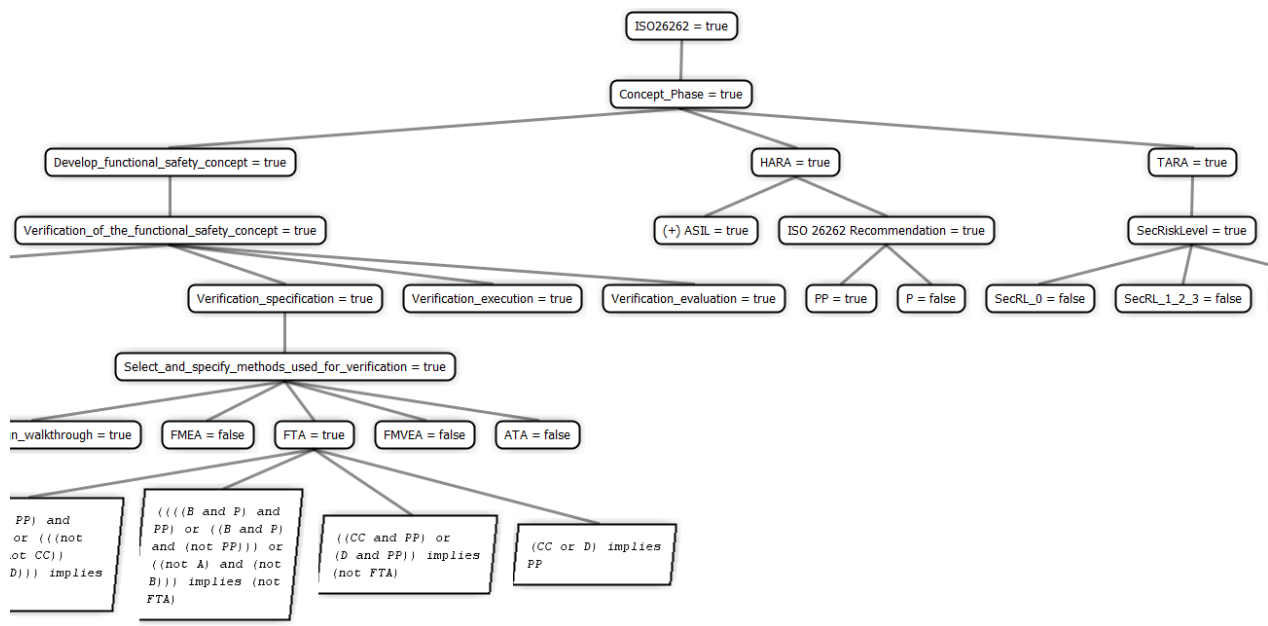


**Figure 33.** VSpec diagram: Concept phase of an integrated safety and security process (ASIL:=B)

The variability choice parameters ASIL, SRL and ReL decide whether specific methods have to be executed or not. The model contains for each level a choice (Rectangles with rounded edges in the diagrams are called “Choice”). In the BVR-tool, constraints decide which activities have to be part of the model. Constraints based on the Basic Constraint Language are used to replicate the standards' varying requirements in the model. Constraints are evaluated if the BVR-function "Validate" is selected. This validation makes sure that the created model complies with the defined constraints. The example shows how constraints are used to deal with a model that combines two standards. From the available ASIL- and ReL-values, one of each has to be selected. The example deals only with ASIL and ReL because the standards do not define the SRL. It has to be defined as project-specific.

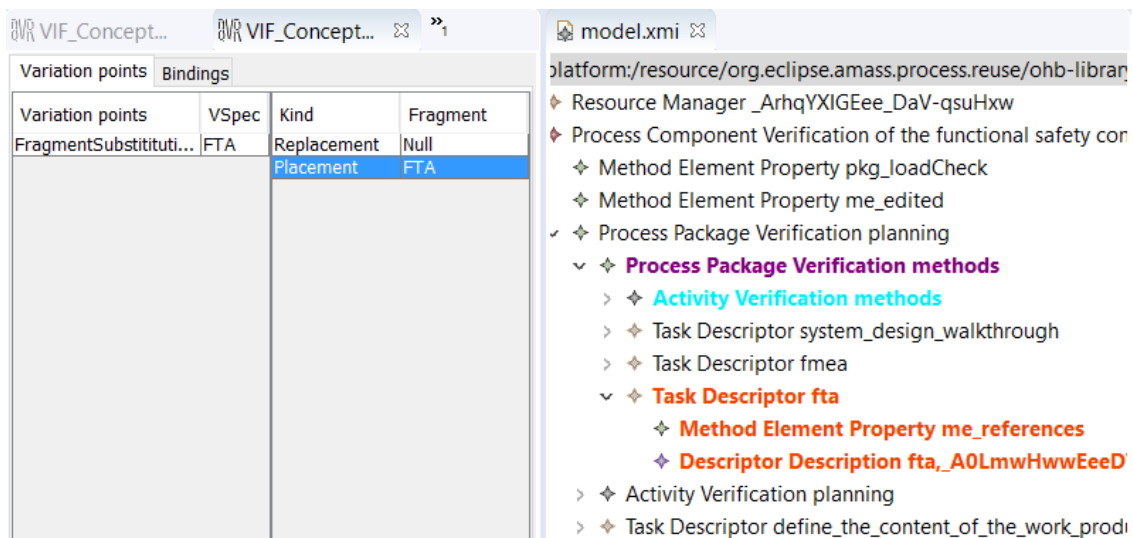
ReL is equivalent to tables from ISO 26262. Figure 31 shows a method table template concerning analysis methods considered in the example. The recommendation level “+” (recommended) is in the example represented by “P” and “++” (highly recommended) is represented by “PP”. With the help of recommendations added to the constraints, process designers have the opportunity to select which recommendation level should be part of the process model. These decisions are based on the company’s guiding principles.

In the BVR-model “ReL” is only linked with ASIL-depending methods because SAE J3061 provides no recommendations in its current release. Generally, security related recommendations can be part of a project specific model and will be a topic for the upcoming automotive security standards.



**Figure 34.** Resolution diagram: Concept phase of an integrated safety and security process (ASIL:=B)

The result of the evaluation of the constraints, done in the “Resolution diagram”, is either "true" or "false". The result “true” means that the “Resolution diagram” is in accordance with its constraints. A choice, an activity in the example, is going to be replaced according to the Realization diagram if its status is “true” (e.g. FTA = true).



**Figure 35.** Realization diagram and imported EPF-C model (Placement for Fragment Substitution in red)

The “Realization diagram” is used to define "Placements" and "Replacements". Detailed description can be found in section 5.3.1 Base Variability Resolution. A placement defines the set of elements that is to be replaced. The replacement is the new set that is put to the placements position. In our example, the replacement is always "null" because the intention is to remove elements from the process (see Figure 35).

The definition of the placement is done in the imported EPF-C model, where we select activities, which are candidates for the replacement (red highlighted elements in Figure 35).

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-” and the “VSpec diagram”. As a



consequence, the placement in the “VSpec diagram” is replaced with "null" if the boolean value of the element defined in the "FragmentSubstitution" is "true".

The last step is the export of the changed process model. This means that we use the "execution" function of the “Resolution diagram” to export the final process model back to an EPF-C processable XMI format. Figure 36 shows a comparison between the EPF-C base process model (left side) and the changed BVR export process model (right side).

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

Figure 36. EPF-C model before (left) and after replacement of FTA (right)

Up to now, no requirements concerning security recommendation levels and security risk levels are available in SAE J3061. In the automotive domain, they are only determined by experienced company specific security processes. A proposal for security risk levels in the automotive domain is in elaboration in the security standardization work. Standards for automation in the industrial domain already define security levels and also provide recommendations, e.g. IEC 62443 [101].

The exported realization from the BVR-tool represents an input for WEFACT. WEFACT covers process management and execution.

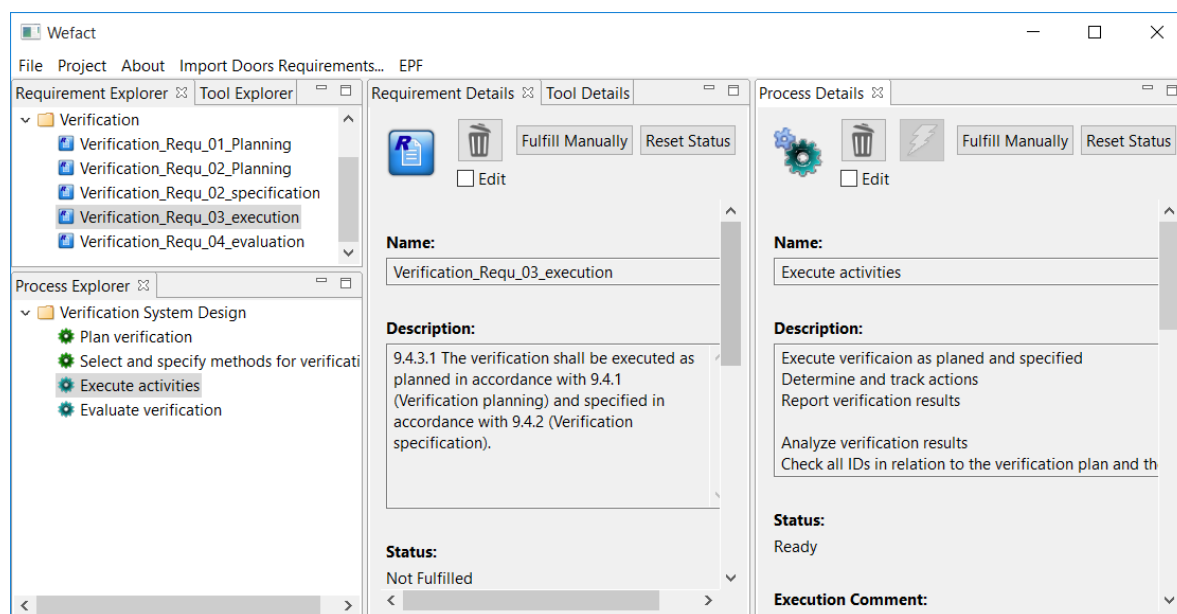


Figure 37. Process for verification in WEFACT

Before the EPF-C model can be executed, it has to 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, workflow tools have to 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.

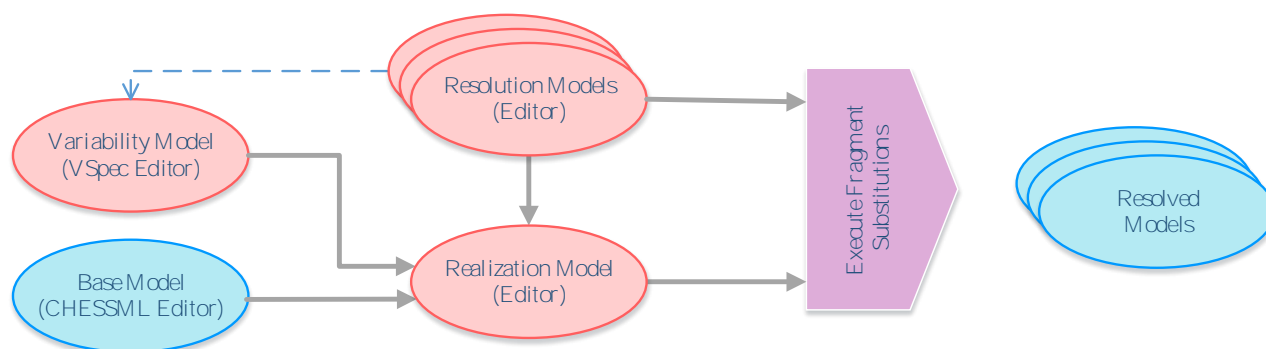
A more detailed description concerning the process execution in WEFACT is available in the deliverable D4.3 [13].

### 5.3.3 Product-related reuse via management of product lines (\*)

To be able to automatically reuse the elements that were presented in 4.2.1, it is necessary to model them with a tool-supported language.

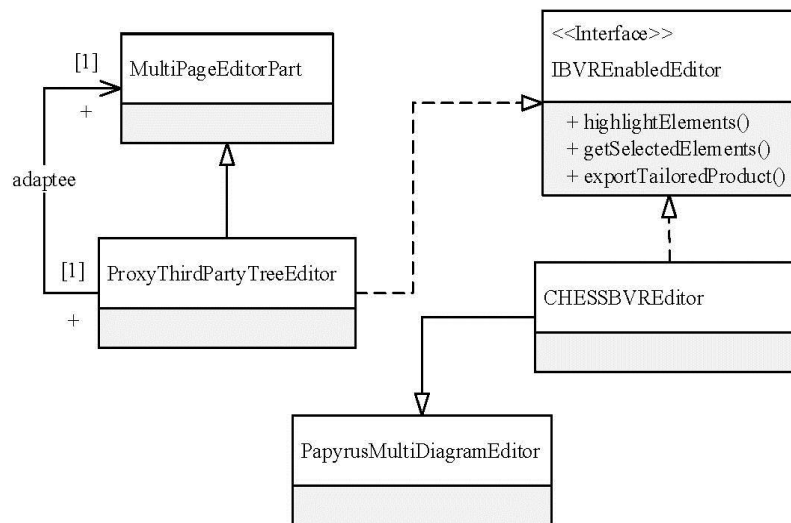
In the context of AMASS, CHESSML has been selected to model the systems. CHESSML, however, does not offer any support for managing variability as needed for our purposes. BVR represents a feasible and technically advantageous solution also in this case. The interplay of the models needed to manage systems variability is similar to the case of process variability management.

Figure 38 depicts the interplay of the different models needed to manage process variability via integration of CHESSML and BVR. In particular, designers jointly with variability engineers should provide four models: a *Base Model* (a CHESSML compliant model) regarding the single product (component model); a *VSpec Model* to model the feature diagram associated to the component model, a *Resolution Model* to model the component configuration, and a *Realization Model* to model the placements and replacements. The interplay of these models is then processed to produce a resolved model where substitutions have been executed.



**Figure 38.** Models interplay enabling management of product lines

As it can be seen from Figure 39, *IBVREnabledEditor* is expected to interact with the *CHESSBVREditor* in order to highlight/select modelling elements to be placed/replaced as well as to export *TailoredProducts*.



**Figure 39.** Architecture of the seamless integrator plugin enabling product-related variability management

### 5.3.3.1 Intra-domain variability management: an aerospace SPL

In this subsection, as a running example, the attitude orbit controller system (part of AMASS CS11 and extensively described in [142]) is used. Thus, first some essential information is recalled. Then, the system is modelled using CHESSEML. Finally, the model is used as Base Model within BVR Tool.

#### 5.3.3.1.1 Running Use Case: Attitude Control Systems (ACSs), taken from [142]

Attitude Control Systems (ACSs) play an important role within satellites. More specifically, ACSs contribute to maintaining a certain attitude (i.e., orientation of the satellite in three-dimensional space). Controlling the attitude is necessary to enable satellites (spacecrafts, which, typically, orbit earth) to accomplish their mission including the fulfilment of their pointing requirements, often referred to as pointing modes. ACSs control the satellite's attitude. This control-related functionality is relative to a frame of reference depending upon the pointing requirement, which is either mission mode or safe-hold mode. The former refers to the main objective of the satellite and latter contributes to fail safe. For instance, a sun pointing mode could be a safe hold mode for a satellite pointing its solar arrays towards the sun to power the critical parts of the satellite. In sun pointing mode, the attitude of a satellite, relative to the sun, is maintained by controlling the torques applied to the satellite by actuator thrusters. A sun sensor measures the angle of the sun beam, in sensor's reference frame, when the beam hits the sensor. The direction of the sun is acquired from this angle and fed into the control system as an input. The control system calculates the torque based on minimizing the pointing error function.

Typically, an ACS (focus on software) is a composite component composed of the following four software components: (1) *PDController* takes the Sun direction and angular velocity of the satellite in three axes as an input, computes the pointing error and calculates proportional and derivative torques; (2) *SteerController* also computes the proportional torque by minimizing the pointing error. But, the objective is to compute relatively greater torques for faster convergence to target pointing position. This is important in cases when the satellite is significantly diverging from the desired position e.g., when it is upside down; (3) *FeedforwardController* computes additional torques to more quickly achieve the equilibrium state and stabilize the satellite body. A satellite in the space is subject to disturbance torques coming from outside of the boundary of the satellite itself. For example, the Sun radiation pressure, the gravitational pull of the celestial bodies, the earth atmospheric pressure, etc. *FeedforwardController* takes these torques as an input and calculates a complementary torque to counteract the disturbance. Typically, these disturbance-related torques are calculated by the angular rates of the satellite, as measured a gyroscope sensor; and (4) *TorqueSelector* takes the torques computed by the above-mentioned controllers, and, based on the current attitude of the satellite, allows a specific torque to be applied on the satellite through the thrusters. If the satellite is oriented 180 degrees opposite to the desired pointing direction, a rapid change in attitude is required. Hence, relatively



greater torques, which are computed by SteerController, are applied. On the other hand, when the satellite is closer to the desired pointing direction, rather a seamless convergence and stabilization is of interest. Therefore, smaller torques and stabilization torques, computed by PDController and FeedforwController respectively, are applied to satellite.

### 5.3.3.1.2 Modelling of ACS in CHESSML

Figure 40 shows the composite given in CHESSML. This composite represents the ACS.

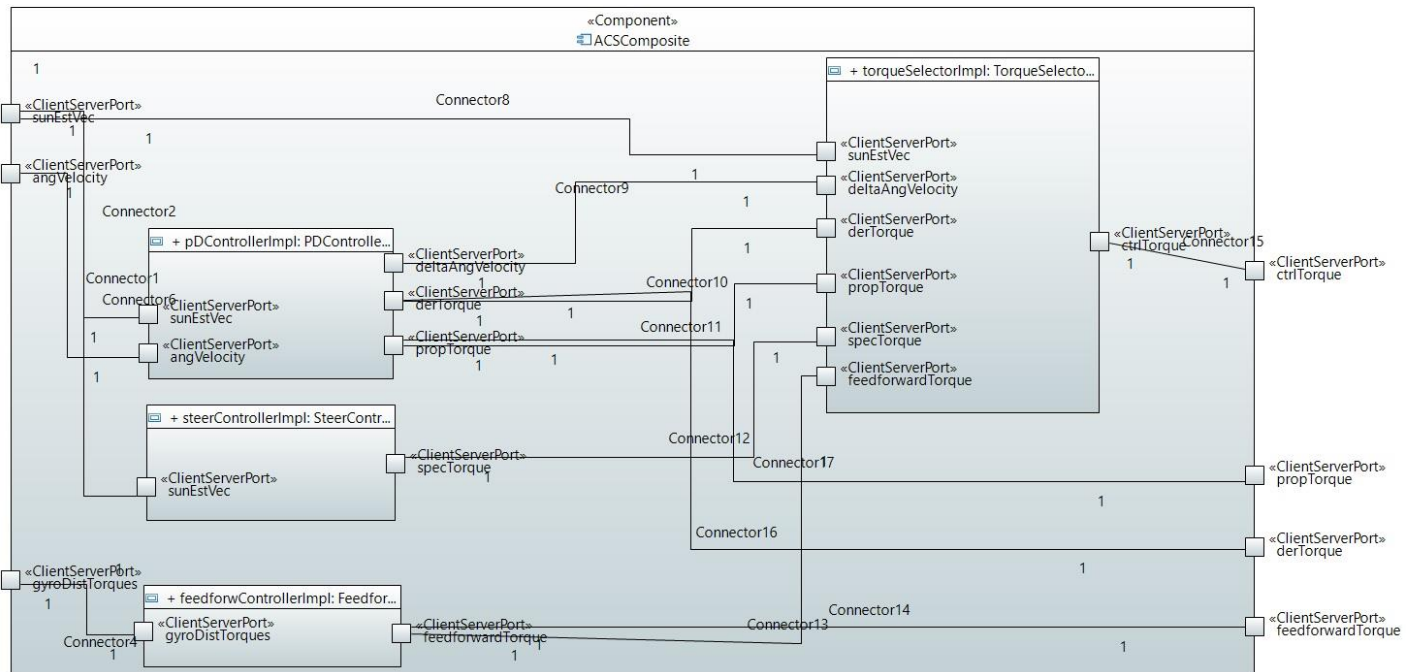
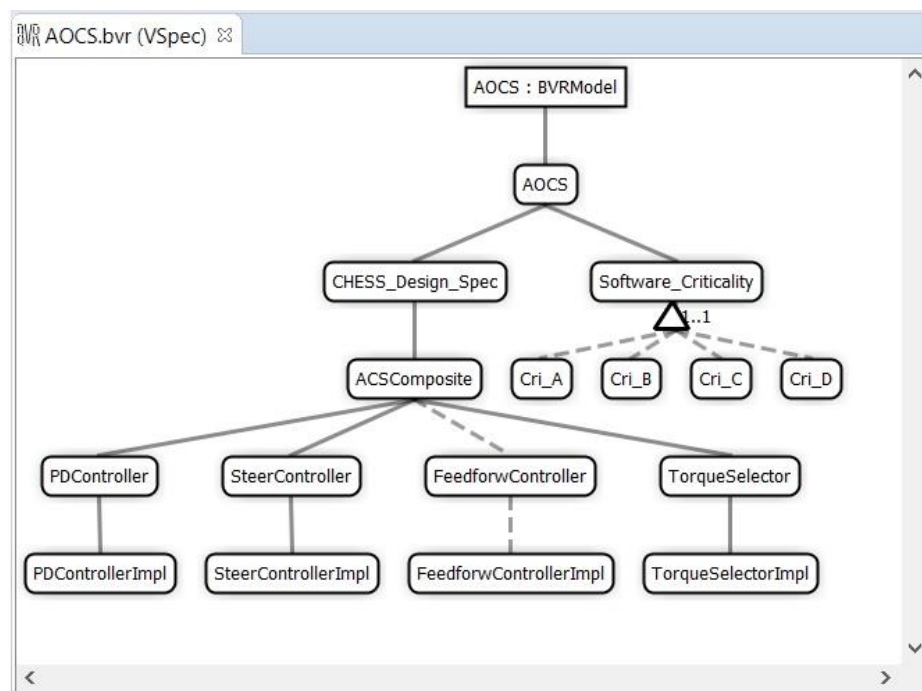
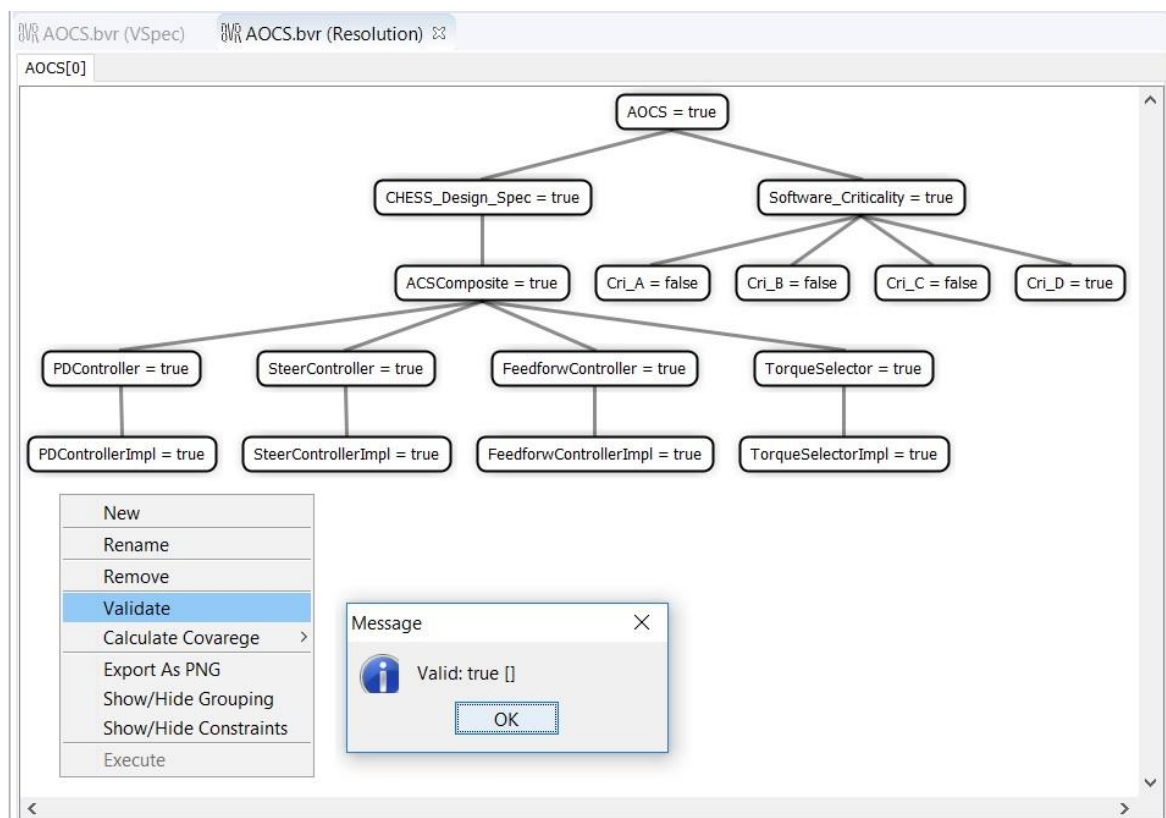


Figure 40. ACS component model given in CHESSML

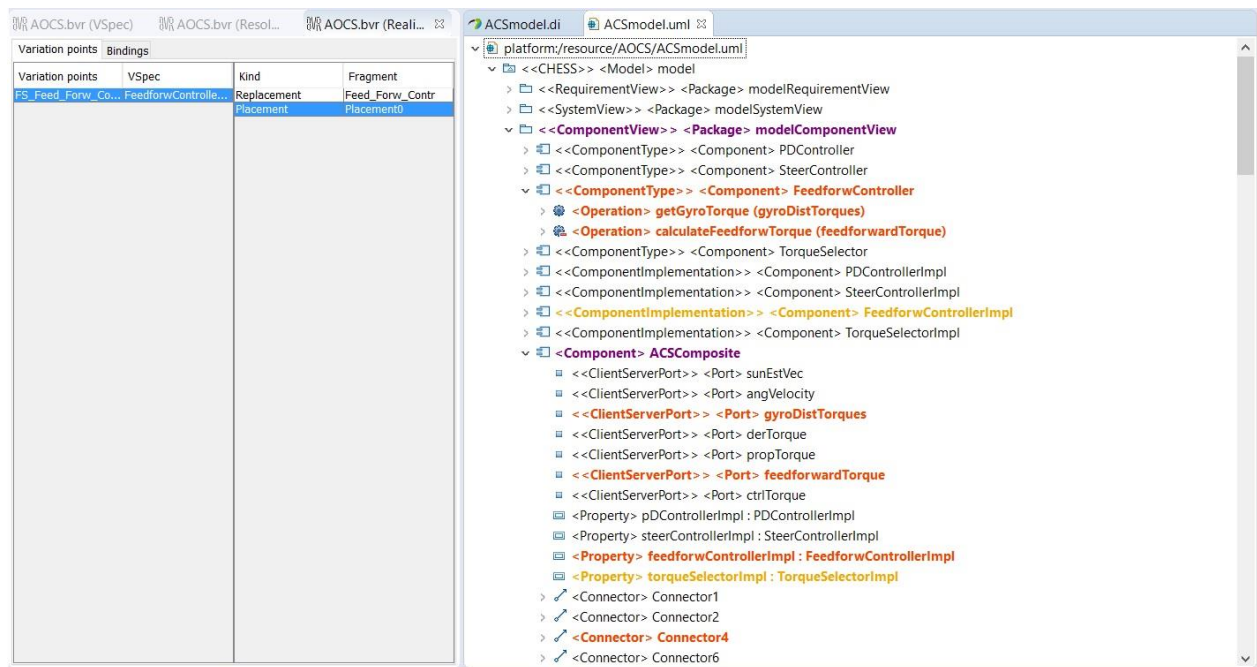
Figure 41, Figure 42, and Figure 43 show respectively: the VSpec model, the Resolution model, and the Realization model regarding the Attitude Control System (ACS), part of the Attitude Orbit Control System.



**Figure 41.** VSpec Model regarding Attitude Control System (ACS)



**Figure 42.** Resolution Model regarding Attitude Control System (ACS)

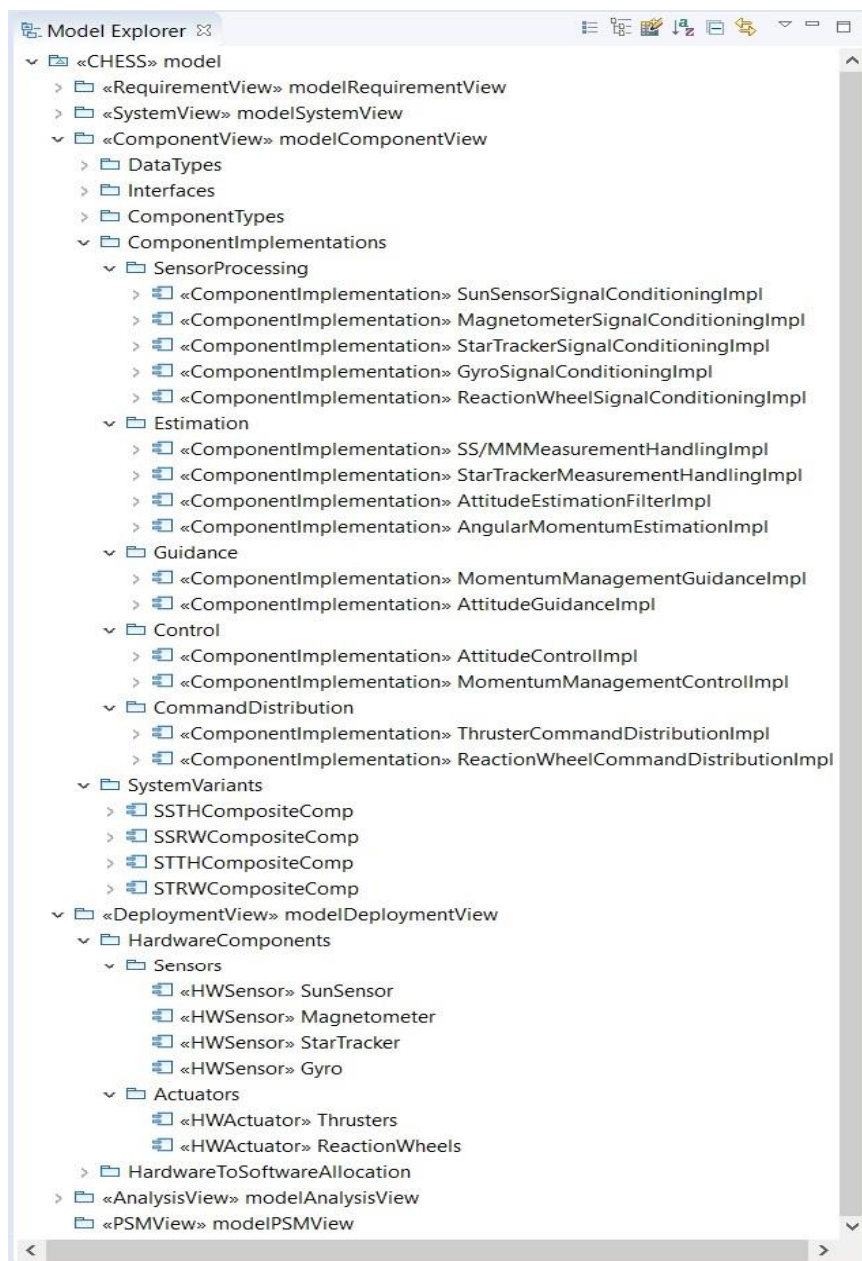


**Figure 43.** Realization Model regarding Attitude Control System (ACS)

#### 5.3.3.1.3 Running Use Case: Attitude Orbit Control Systems (AOCSs) -extension (\*)

This section focuses on the family of AOCSs. The family presented in this section represents a still simplified but richer example with respect to the one presented in Section 5.3.3.1.1. This family is obtained by making vary one family member, initially modeled in CHESSML. Figure 44 shows the modelling of the different views (requirement, system, component, deployment, analysis, and PSM views), created for one AOCS.





**Figure 44.** CHES Views

These views constitute base models, which can be made vary via the integration with BVR Tool in order to obtain four different variants/configurations: (i) Sun Sensors THrusters (SSTH); (ii) Sun Sensors Reaction Wheels (SSRW); (iii) Star Tracker THrusters (STTH); and (iv) Star Tracker Reaction Wheels (STRW).

These variants (which focus on the component view) are illustrated in Figure 45-48 (note that expanded version of these figures are available in Appendix B). These variants are configured based on the different allowed combinations of: sensors, actuators, as well as the associated functional software modules. These combinations are informally specified in Figure 49.

In the context of sensing principles, Sun Sensors are used as primary sensors in combination with magnetometer, or otherwise Star Tracker is used as a primary attitude sensor. The Sun Sensors, together with magnetometer, are used in SSTH and SSRW variants to provide full attitude determination capability. However, the STTH and STRW variants focus on Star Tracker which itself is able to provide 3-axis attitude measurements. In all variants, the attitude is estimated using a kinematic (Kalman) attitude estimation filter for which Gyro measurements are an exogenous input.





In the context of actuation principles, the spacecraft attitude is modified using a Thruster actuation system; otherwise, the attitude is primarily controlled using Reaction Wheels. In STH and STTH variants, the spacecraft attitude is modified using a Thruster actuation system, which is able to provide a control torque in any direction. This results in a zero-momentum system. In SSRW and STRW variants, the spacecraft attitude is primarily controlled using Reaction Wheels. These are momentum storage devices (3 rotating masses in e.g. a perpendicular configuration) that are able to provide a reaction torque in any direction. Since the reaction wheels are not able to alter the system's angular momentum, Thrusters are occasionally used to reduce the system angular momentum in order to keep this bounded over long term.

The functional software modules are classified into different functional groups: 1) sensor processing (responsible of converting raw sensor measurements into engineering values expressed in spacecraft coordinates); 2) estimation (responsible of estimating physical quantities such as spacecraft attitude, rate and the angular momentum); 3) guidance (responsible of providing the spacecraft attitude and angular momentum as commanded from the ground); 4) control (responsible of controlling attitude and angular momentum); and 5) command distribution (responsible of converting the desired control torques expressed in spacecraft coordinates to individual raw actuator commands). In general, the functional software modules are architecturally and functionally the same when used in different variants, even though they will need dedicated tuning (values of their parameterization) for each variant.

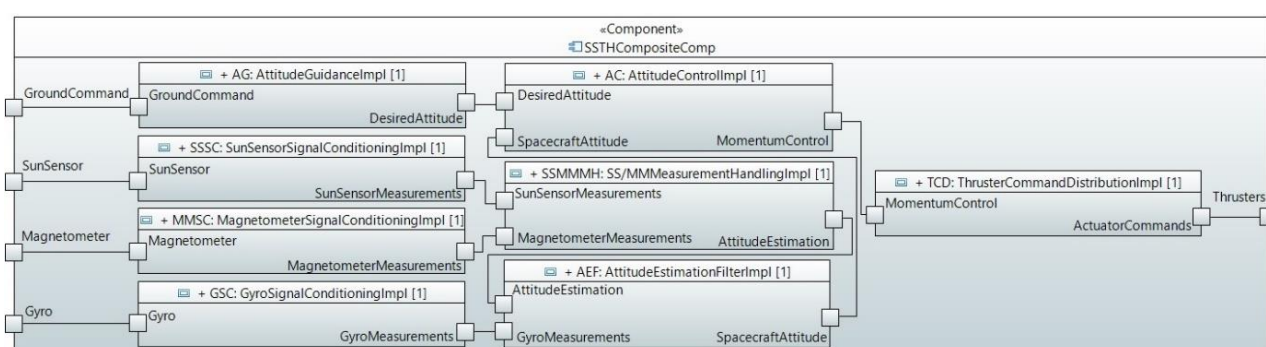


Figure 45. STH (Sun Sensors Thrusters)

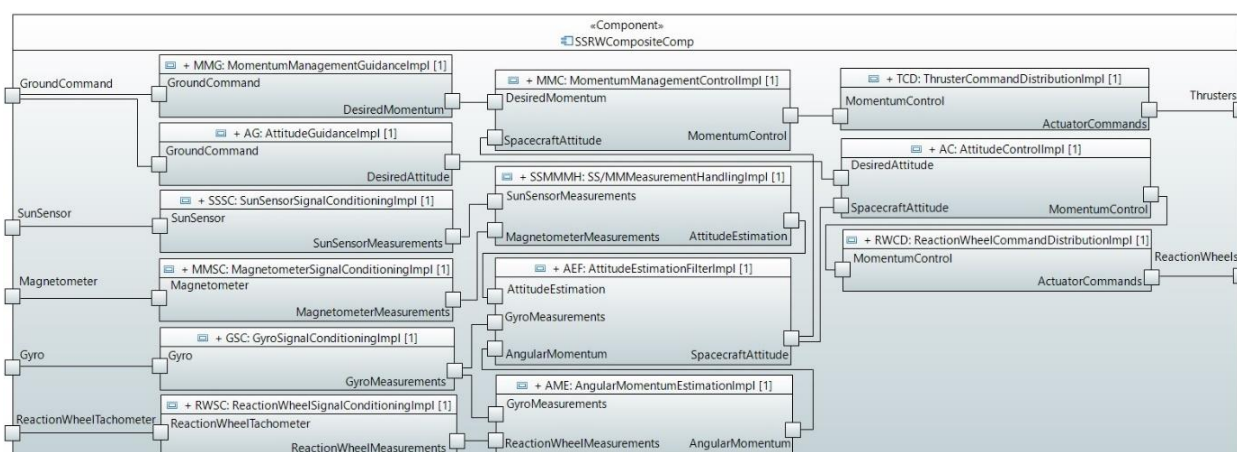


Figure 46. SSRW (Sun Sensors Reaction Wheels)

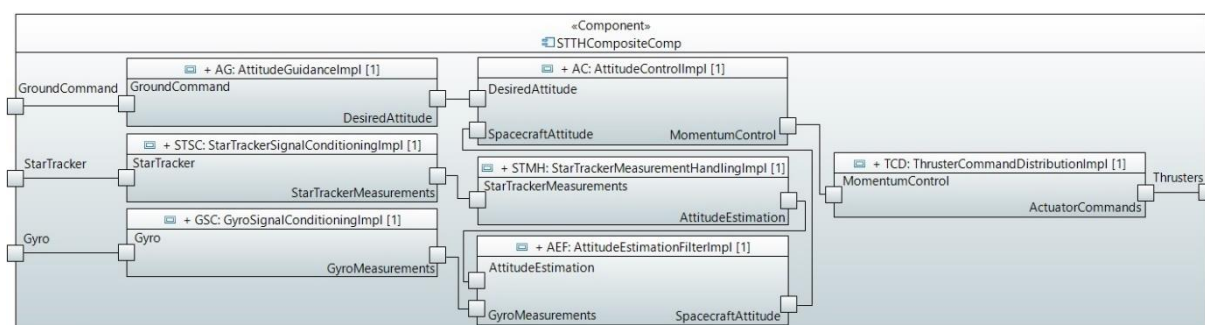


Figure 47. STTH (Star Tracker Thrusters)

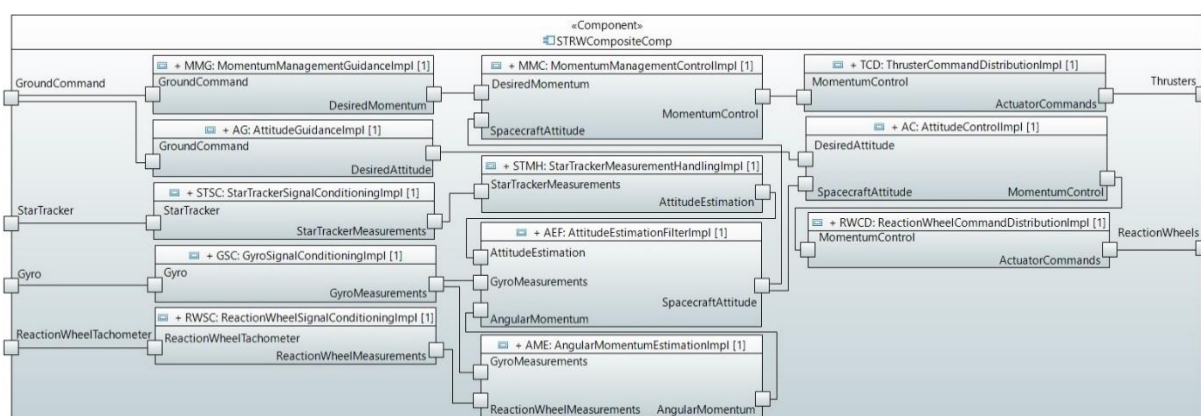


Figure 48. STRW (Star Tracker Reaction Wheels)



	1: SSTH	2: SSRW	3: STTH	4: STRW
<b>Sensors</b>				
Sun Sensor	X	X		
Magnetometer	X	X		
Star Tracker			X	X
Gyro	X	X	X	X
<b>Sensor Processing</b>				
Sun Sensor Signal Conditioning	X	X		
Magnetometer Signal Conditioning	X	X		
Star Tracker Signal Conditioning			X	X
Gyro Signal Conditioning	X	X	X	X
Reaction Wheel Signal Conditioning		X		X
<b>Estimation</b>				
Sun Sensor / Magnetometer Measurement Handling	X	X		
Star Tracker Measurement Handling			X	X
Attitude Estimation Filter	X	X	X	X
Angular Momentum Estimation		X		X
<b>Guidance</b>				
Attitude Guidance	X	X	X	X
Momentum Management Guidance		X		X
<b>Control</b>				
Attitude Control	X	X	X	X
Momentum Management Control		X		X
<b>Command Distribution</b>				
Thruster Command Distribution	X	X	X	X
Reaction Wheel Command Distribution		X		X
<b>Actuators</b>				
Thrusters	X	X	X	X
Reaction Wheels		X		X

Figure 49. SSTH, SSRW, STTH and STRW Configurations

### Establishment of Intra-Domain Product Line

This section focuses on the engineering of AOCSs-family (product line) via the exploitation of the seamless integration between CHES Tool and Base Variability Resolution (BVR) Tool, which was explained in Section 5.3.3.

Based on Figure 49, AOCS-family is informally engineered. More specifically, the domain engineering: identification of commonalities and variabilities to concurrently engineer a set of products is done by observing which actuators/sensors/functional software modules have to be present in all AOCS and which ones instead only in some of them.

The achievement of single AOCSs, application engineering, is based on the selection and composition of commonalities and variabilities. To enable the engineering of the AOCS-family systemically, BVR Tool is used.

The generation of the targeted configurations for the AOCS variants modelled in CHESML are performed with VSpec, Resolution, and Realization editors.

As explained in Section 5.3.3, a VSpec model has to be created. To do that, the sensors, actuators and associated software functional modules are modeled with the VSpec editor. The obtained VSpec model, shown in Figure 50, shows the tree structure representing the systematization of the commonalities and the variability within a formal. The logical constraints, which define cross-cutting inclusion/exclusion dependencies, can be automatically checked during the resolution, within the Resolution editor.



Once again it is recalled that solid lines indicate that the particular feature applies to all variants (e.g., Thrusters, under actuators), whereas the dashed lines represent the variation points. The whole tree cannot be visualized due to space limitations; therefore, the minimize option (+) is used for hiding the features. The varying modules are marked as optional. The constraints have been applied, in which logical operators such as implication, alternative, negation might be used. Thus, valid tailoring is guaranteed if the constraints are properly specified. For instance, the constraint “SunSensor implies (Magnetometer and (not StarTracker))” (modelled in Figure 50) enforces inclusion of Magnetometer, but also exclusion of StarTracker.

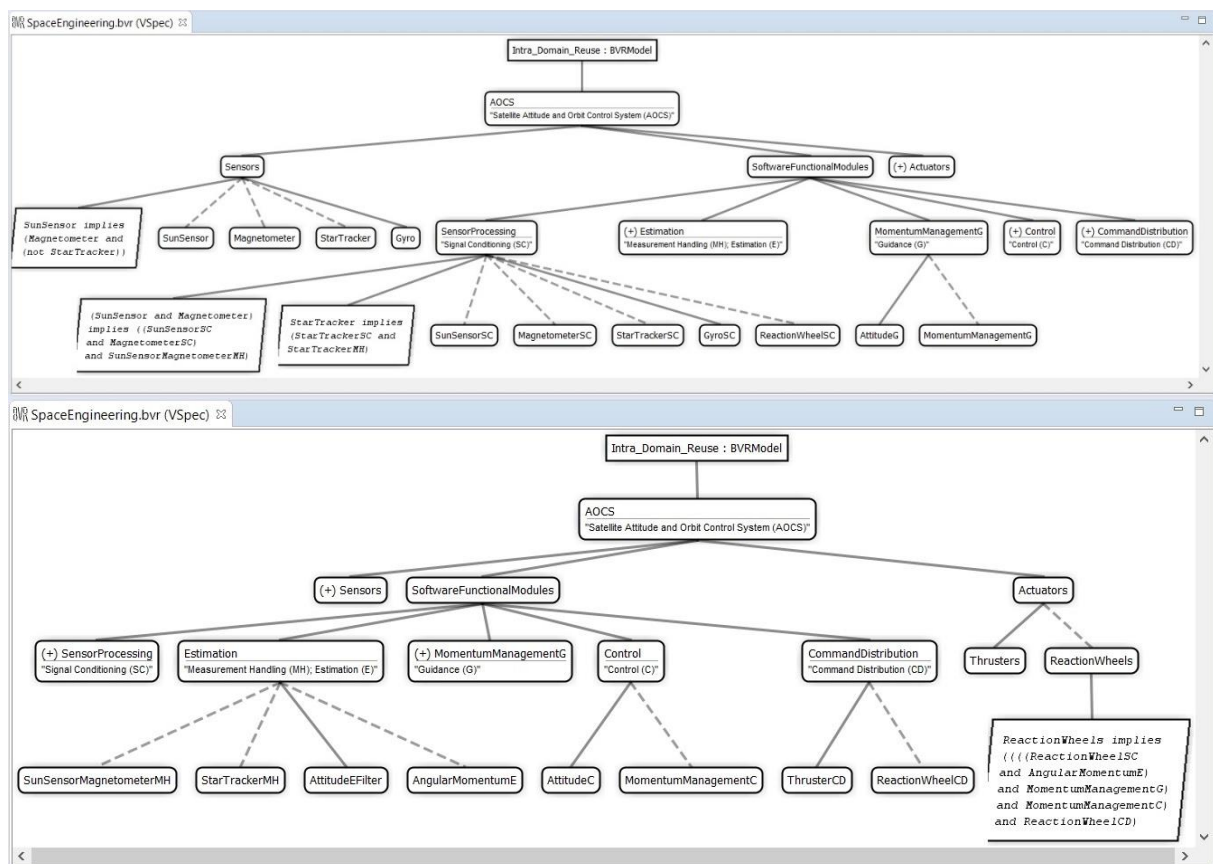


Figure 50. Intra-Domain Variability Modelling

The resolution models are automatically generated from the VSpec model except for the choices (expressing what needs to be included or excluded for individual products). The user takes the decisions at the variation points (by setting true to the feature to be included), as shown in Figure 51. Once a resolution model is finalized it can be checked for consistency to ensure that the configuration is allowed.



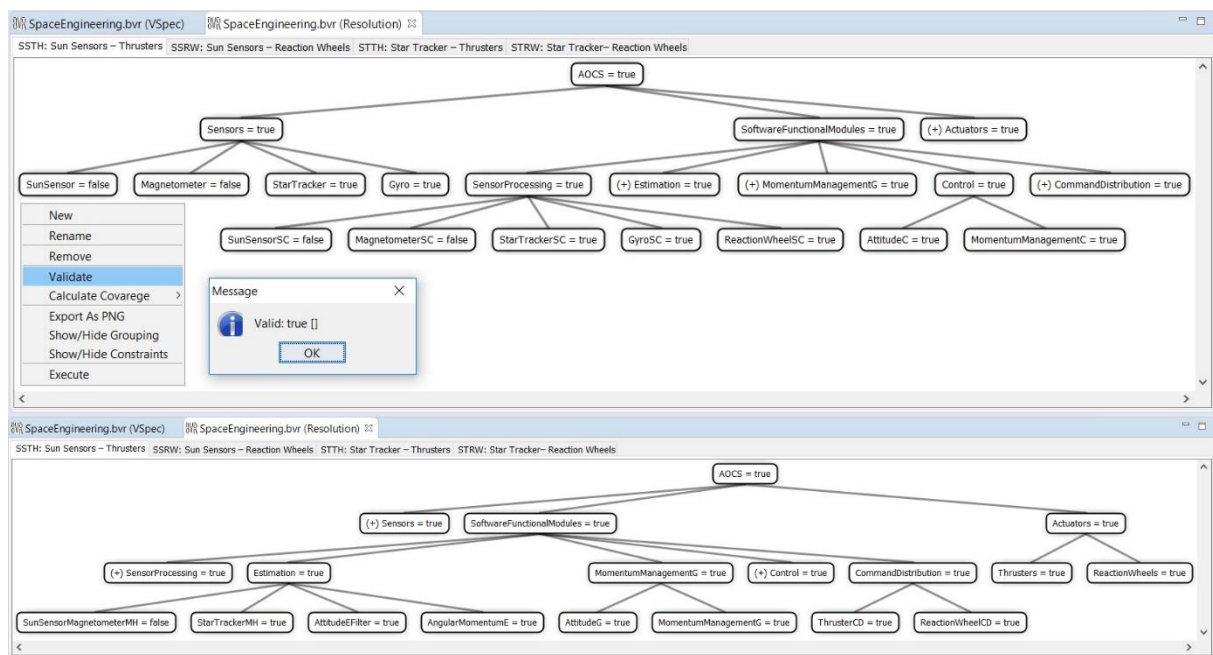


Figure 51. Intra-Domain Variability Resolution

At this point, everything that is needed, to make vary a CHESML-compliant component model representing a specific AOCs, is at disposal. The final step is the realization, which involves the substitutions (in which elements of a placement fragment are removed and elements of a replacement are injected), as shown in Figure 52.

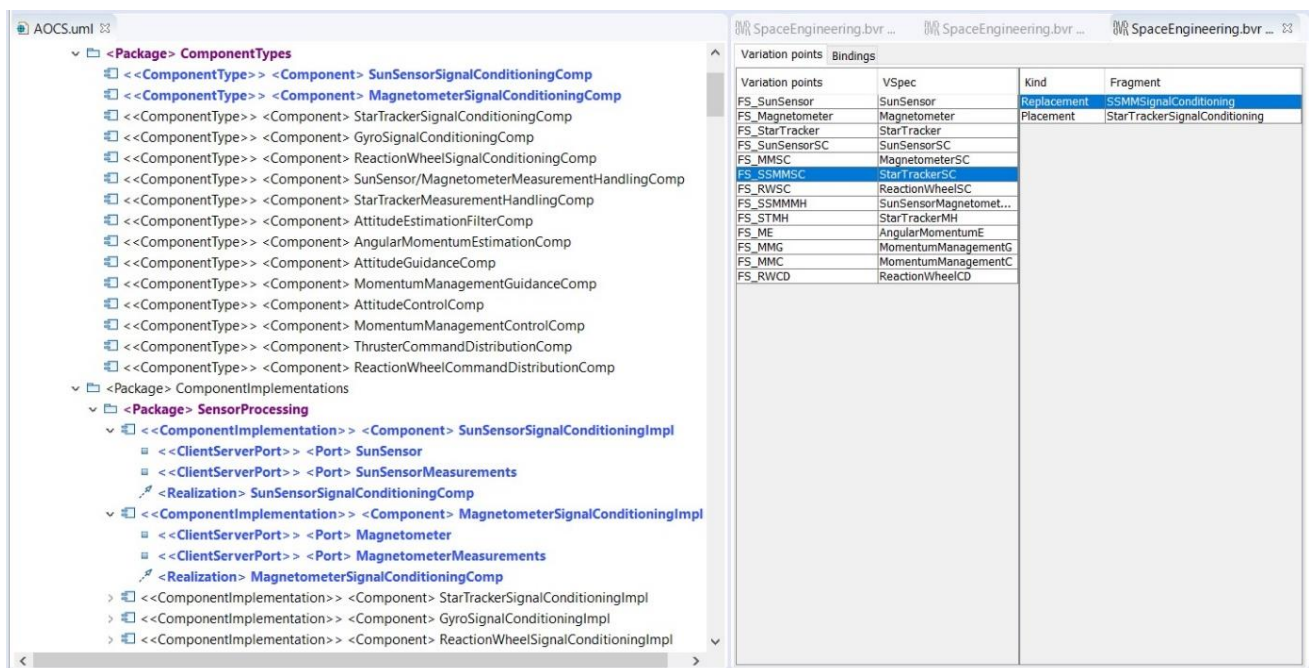


Figure 52. Intra-Domain Variability Realization

### 5.3.3.2 Cross-domain variability management: an automotive/avionics product line (\*)

In this section, a cross-domain product line is engineered from two different domain-specific products (an automotive unit and an avionics unit). First the automotive unit by Infineon is textually described; then the avionics unit by Lange is textually described. Finally, CHESML and BVR are used to model the cross-domain product line and show how a CHESML-compliant component model representing the architecture of one unit



can be made vary via BVR Tool to obtain the CHESML-compliant component model representing the architecture of the other unit.

The purpose of this section is not to engineer an in-depth cross-domain product line constituted of central and electronic controlling/computing units. The purpose is limited to illustrate how the designed solution could serve the purpose. A more in-depth engineering is expected to be documented within WP1 deliverables.

#### **5.3.3.2.1 Automotive Electronic Computing Unit**

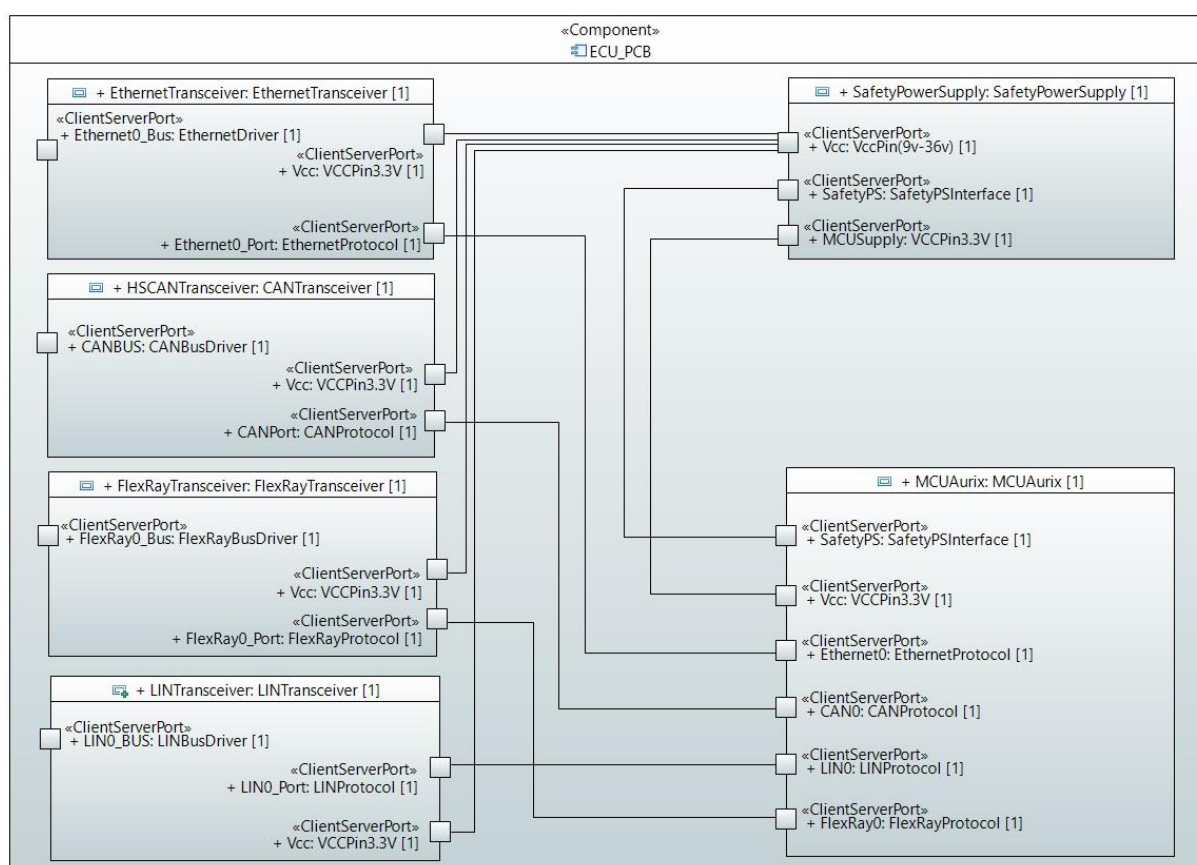
As depicted in Figure 53, Infineon automotive Electronic Control Unit (ECU) model is composed of six components: Ethernet Transceiver, FlexRay Transceiver, CAN Transceiver, Safety Power Supply and AURIX Microcontroller (e.g. AURIX TC39x). These components are briefly explained below:

1. Ethernet Transceiver
  - Interface between the physical bus layer and the Ethernet protocol controller, drives the signals to the bus and protects the microcontroller against interferences generated within the network
  - Qualified for -40°C to +125°C ambient operating temperature
2. FlexRay Transceiver
  - Interface between the physical bus layer and the Ethernet protocol controller, drives the signals to the bus and protects the microcontroller against interferences generated within the network
  - FlexRay Electrical Physical Layer Specification, version 3.0.1 and ISO 17458
  - Optimized for time-triggered in-vehicle networks with data transmission rates from 1 Mbit/s up to 10 Mbit/s
  - Very low electromagnetic emission (EME), supporting large networks and complex bus topologies
  - Very high level of ESD robustness, 11 kV according to IEC-61000-4-2
  - Automatic voltage adaptation on the digital interface pins
  - High current digital outputs, optimized to drive long wires and high capacitive loads
  - Qualified for -40°C to +125°C ambient operating temperature
  - Digital I/O levels compatible with 3.3 V and 5 V microcontrollers
3. LIN Transceiver
  - Interface between the physical bus layer and the Ethernet protocol controller, drives the signals to the bus and protects the microcontroller against interferences generated within the network
  - Single-wire LIN transceiver for transmission rates up to 20 kbps
  - Compliant to ISO 17987-4, LIN Specification 2.2A and SAE J2602
  - Very low current consumption in Sleep mode with wake-up capability
  - Over temperature protection and supply under voltage detection
  - Very high ESD robustness,  $\pm 10$  kV according to IEC61000-4-2
  - Optimized for high electromagnetic compatibility (EMC)
  - Very low emission and high immunity to interference
  - The TLE7258 operate as a bus driver between the protocol controller and the physical bus of the LIN network.
4. CAN Transceiver
  - Interface between the physical bus layer and the Ethernet protocol controller, drives the signals to the bus and protects the microcontroller against interferences generated within the network
  - Fully compliant to ISO11898-2/-5
  - Wide common mode range for electromagnetic immunity (EMI)
  - Very low electromagnetic emission (EME)
  - Excellent ESD robustness





- Guaranteed loop delay symmetry to support CAN FD data frames up to 2 MBit/s
  - V IO input for voltage adaption to the microcontroller supply
  - Extended supply range on V CC and V IO supply
  - CAN short circuit proof to ground, battery and V CC
  - TxD time-out function
  - Low CAN bus leakage current in power-down state
  - Over-temperature protection
  - Protected against automotive transients
  - Stand-by mode with remote wake-up function
  - Wake-up indication on the RxD output
  - Transmitter supply V CC can be turned off in stand-by mode
  - Green Product (RoHS compliant)
  - Two package variants: PG-TSON-8 and PG-DSO-8
  - AEC Qualified
5. Safety Power Supply
- Wide operation range up to 45 V
  - Low dropout voltage
  - Wide temperature range: -40°C up to +150°C
  - Short-circuit protection
  - Reverse polarity protection as option
  - Overload protection
  - Over-temperature protection
6. AURIX Microcontroller (e.g. AURIX TC39x)
- Up to 6 cores
  - Flash memory sizes of up to 16 Mbyte and more than 6 Mbyte integrated RAM
  - Cores with each a full clock frequency of 300 MHz
  - Four of the six cores feature additional lockstep cores,
  - Enabling a new level of ISO26262 functional safe computational power on a single integrated device.
  - Gigabit Ethernet interface



**Figure 53.** Infineon Automotive Electronic Computing Unit Model

### 5.3.3.2.2 Aviation Central Computing Unit

As depicted in Figure 54, Lange aviation Central Computing (CCU) unit model is composed of eight components: SUB-D Power Connector, Safety Power Supply, MCU-Aurix, CAN Transceiver (0, 1, 2), ETH PHY, PCB Heating, CAN SUB-D Connector and ETH SUB-D Connector. These components are explained below:

1. SUB-D Power Connector
  - Aviation-grade certified SUB-D connector, MIL-DTL-24308 certified, -55°C to +125°C
2. Safety Power Supply
  - Accept power supply voltage in the range of 9v-36v DC, compatible to both 14V DC and 28V DC aviation power networks
  - Provides 3.3V power to MCU and all the peripheral devices, provides watchdog interface to Aurix MCU
3. MCU-Aurix
  - Aurix based microcontroller, supporting lockstep core, communication with safety power supply, providing digital I/O (PWM), 3 CAN controllers and 1 Ethernet interface
4. CAN Transceiver 0, 1, 2
  - Automotive CAN transceiver providing CAN 2.0B ISO 11898-2 physical layer
  - Several CAN lines are used to achieve redundant communication to sensors and actuators
5. ETH PHY
  - 100 Base T copper automotive Ethernet Phy according to IEEE802.3
  - Ethernet is used is AFDX connection to the flight control system
6. PCB Heating

- Redundant PCB heating circuit. Uses external 9-36V power supply. Controlled by the digital Output from Aurix MCU (PWM)
  - Allows operation under -40°C (limitation of automotive components) down to -55°C
7. CAN SUB-D Connector
- Aviation-grade certified SUB-D connector, MIL-DTL-24308 certified, -55°C to +125°C
8. ETH SUB-D Connector
- Aviation-grade certified SUB-D connector, MIL-DTL-24308 certified, -55°C to +125°C

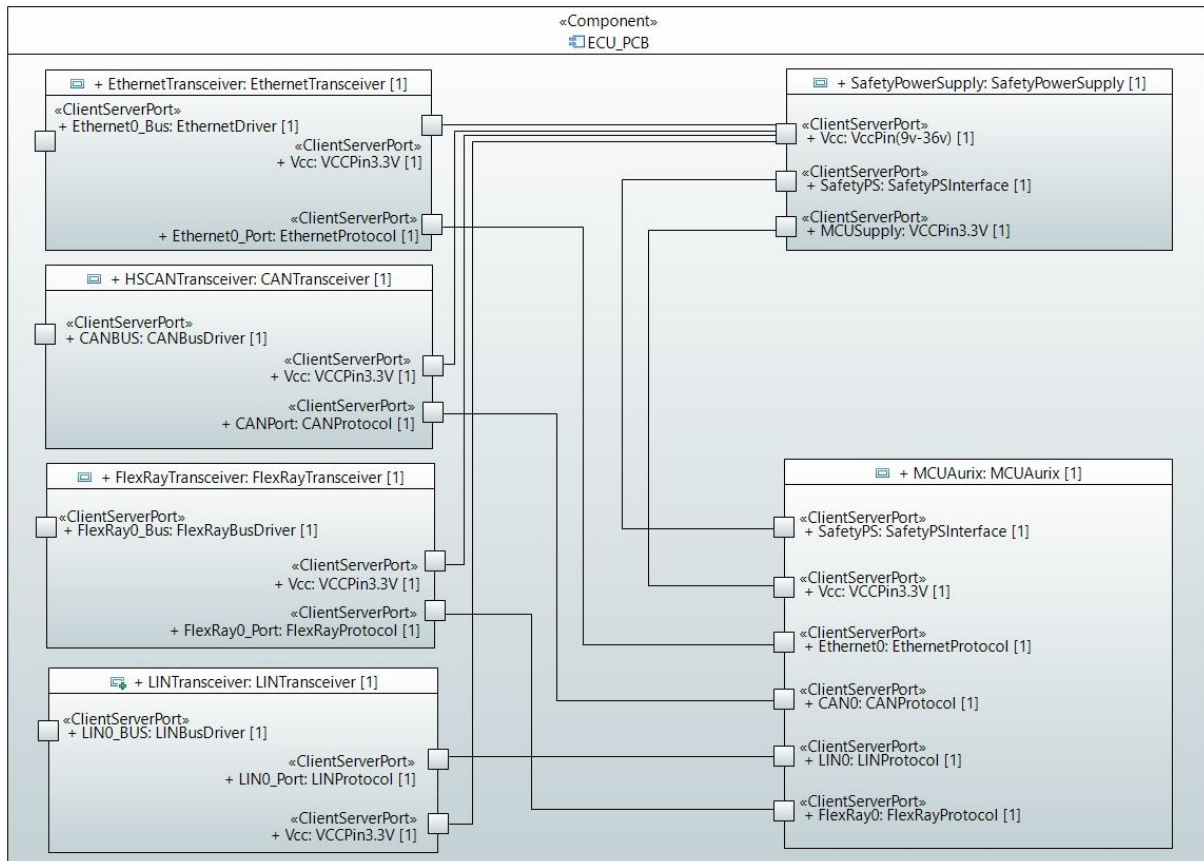


Figure 54. Large Aviation Central Computing Unit Model

### 5.3.3.2.3 Establishment of Cross-Domain Product Line

Reuse is possible, both ECU and CCU share the same HW architecture (which is represented at conceptual level via a VSpec model, see Figure 55. Several components constitute a commonality. However, 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 no reuse of specified components is possible. The connectors of ECU shall also be replaced with the aviation grade SUB-D connectors in CCU.

Similar to the intra-domain product line configuration, the engineering of the targeted configurations for cross-domain is performed with VSpec, Resolution, and Realization editors, as shown in Figure 55-57. For instance, the configuration of the ECU is obtained via choosing the avionics-specific components (e.g., FlexRay and LIN



transceivers) in Figure 56 and then in the Realization editor by substituting the fragments. For instance, Figure 57, the fragment representing the avionics-specific connector (SUB-D Power Connector) is removed.

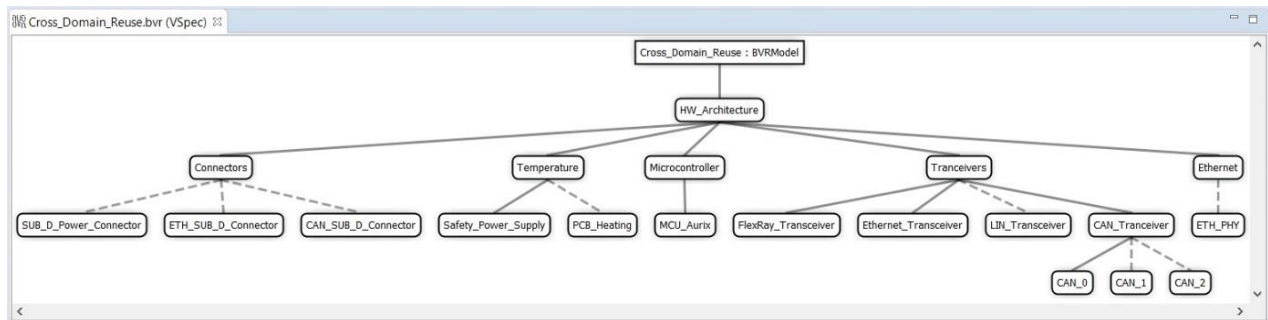


Figure 55. VSpec representing the cross-domain product line

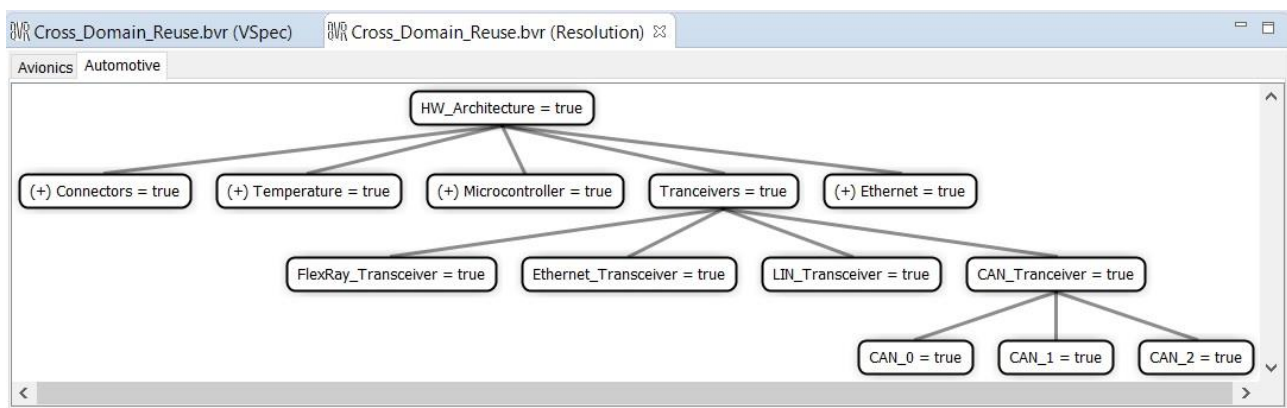


Figure 56. Variability Resolution for Automotive

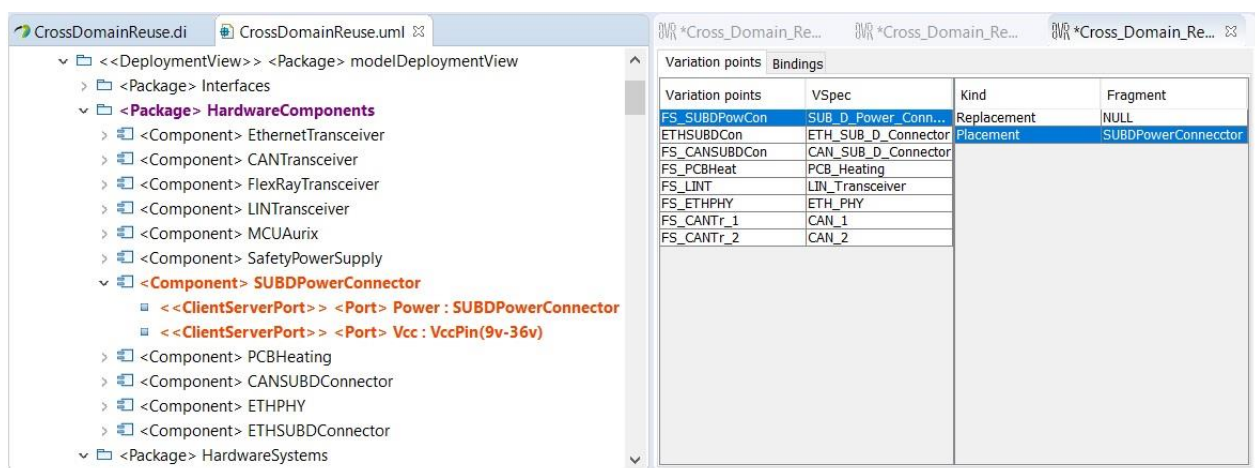
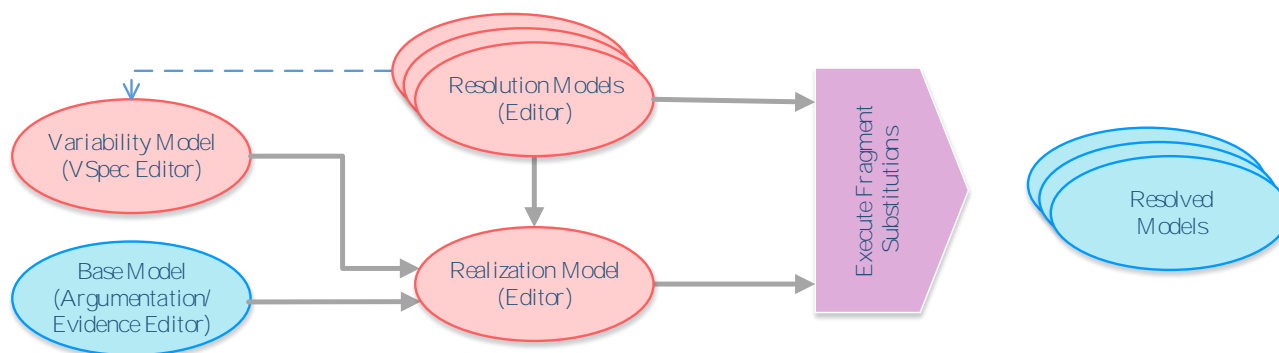


Figure 57. Automotive ECU realization

### 5.3.4 Assurance case-related reuse via management of case lines (\*)

To be able to automatically reuse the elements that were presented in 4.3.1, it is necessary to model them with a tool-supported language. In the context of AMASS, a SACM2.0-like metamodel has been selected to model the assurance cases. This metamodel, however, does not offer any support for managing variability as needed for our purposes. BVR represents a feasible and technically advantageous solution also in this case.

Figure 58 depicts the interplay of the different models needed to manage process variability via integration of the SACM2.0-like metamodel, i.e., CACM (portion focused on Argumentation), and BVR. In particular, assurance managers jointly with variability engineers should provide four models: a *Base Model* (a CACM (focus on argumentation)-compliant model) regarding the single assurance case; a *VSpec Model* to model the feature diagram associated to the assurance case model; a *Resolution Model* to model the assurance case configuration; and a *Realization Model* to model the placements and replacements. The interplay of these models is then processed/elaborated to produce a resolved model where substitutions have been executed.



**Figure 58.** Models interplay enabling management of assurance case lines

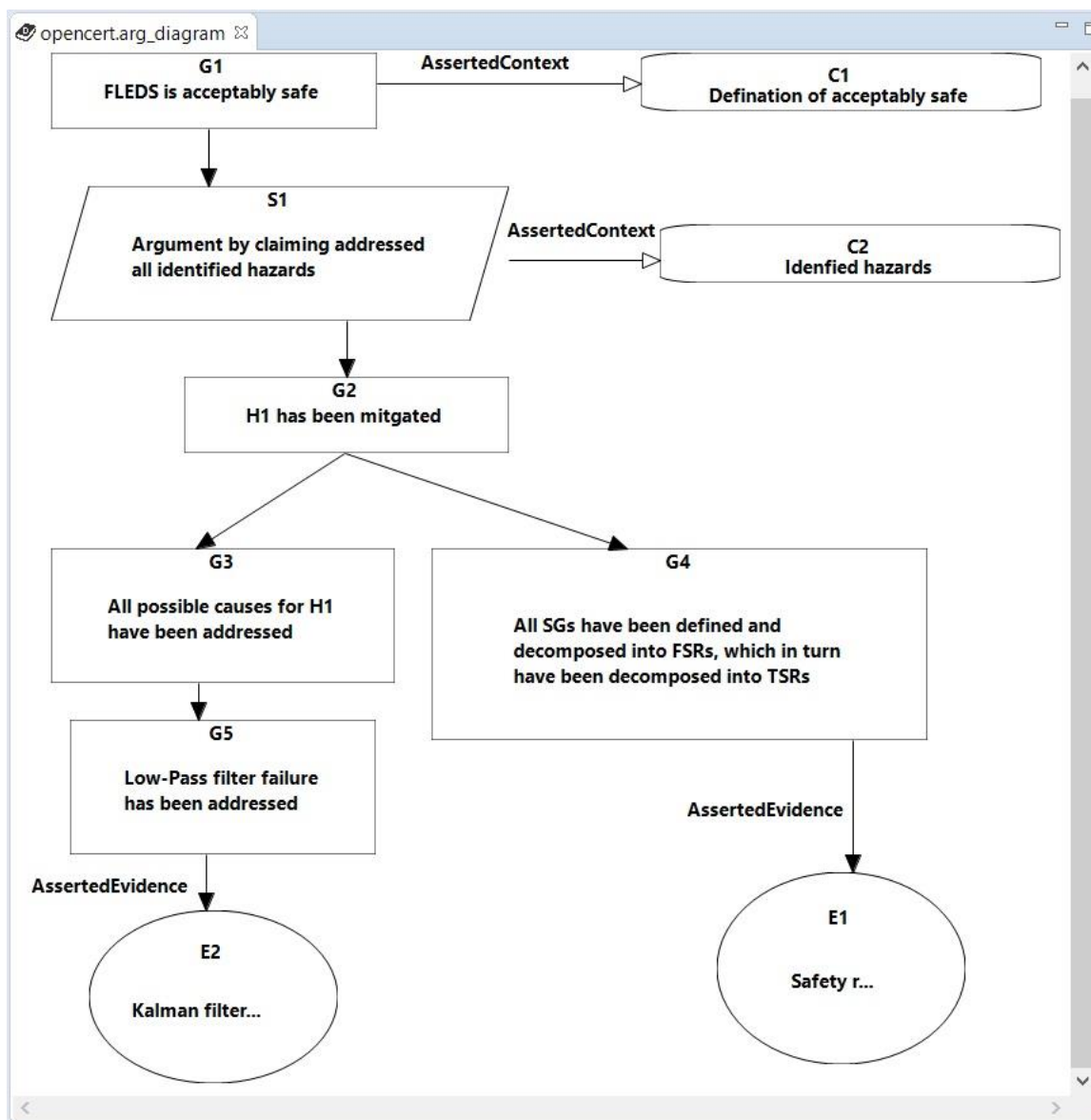
#### 5.3.4.1 Intra-domain variability management at assurance case level: an automotive assurance case line (\*)

In this section, the aim is not to present the management of the variability of a complete automotive safety case line, but only a simple example to illustrate the solution. Thus, first an example is presented and then it is made vary by using the integration of BVR Tool and OpenCert.

In particular, the example consists of a fragment of an automotive safety case, adapted from [158]. This fragment is presented in Figure 59. More specifically, Figure 59 shows a fragment modelled by using the assurance case editor of OpenCert. The argument fragment starts by instantiating the hazard avoidance pattern. It thus claims (G1) that each FLEDS (Fuel Level Estimation and Display System) is acceptably safe, given a definition of “acceptably safe”. G1 is then refined into G2, by using the strategy S1 (i.e. by arguing and claiming that all the identified hazards have been addressed). G2 is then broken down into G3 and G4. G4 is directly supported by the solution E1. G3 is refined into G5, which is supported by direct evidence E2.

In [158], G3 is instead broken down either into G5 or into G6, depending on the filter type (Kalman or not), which in turn depends on the usage context (vehicle type) variation point (where a constraint constrains that in case of vehicle=truck, G5 has to be selected as sub-argument). Finally, G5 and G6 are supported by direct evidence (E2 and E3). In [158], a dialect of GSN was used to represent a family of safety cases (constituted of two possible configurations).

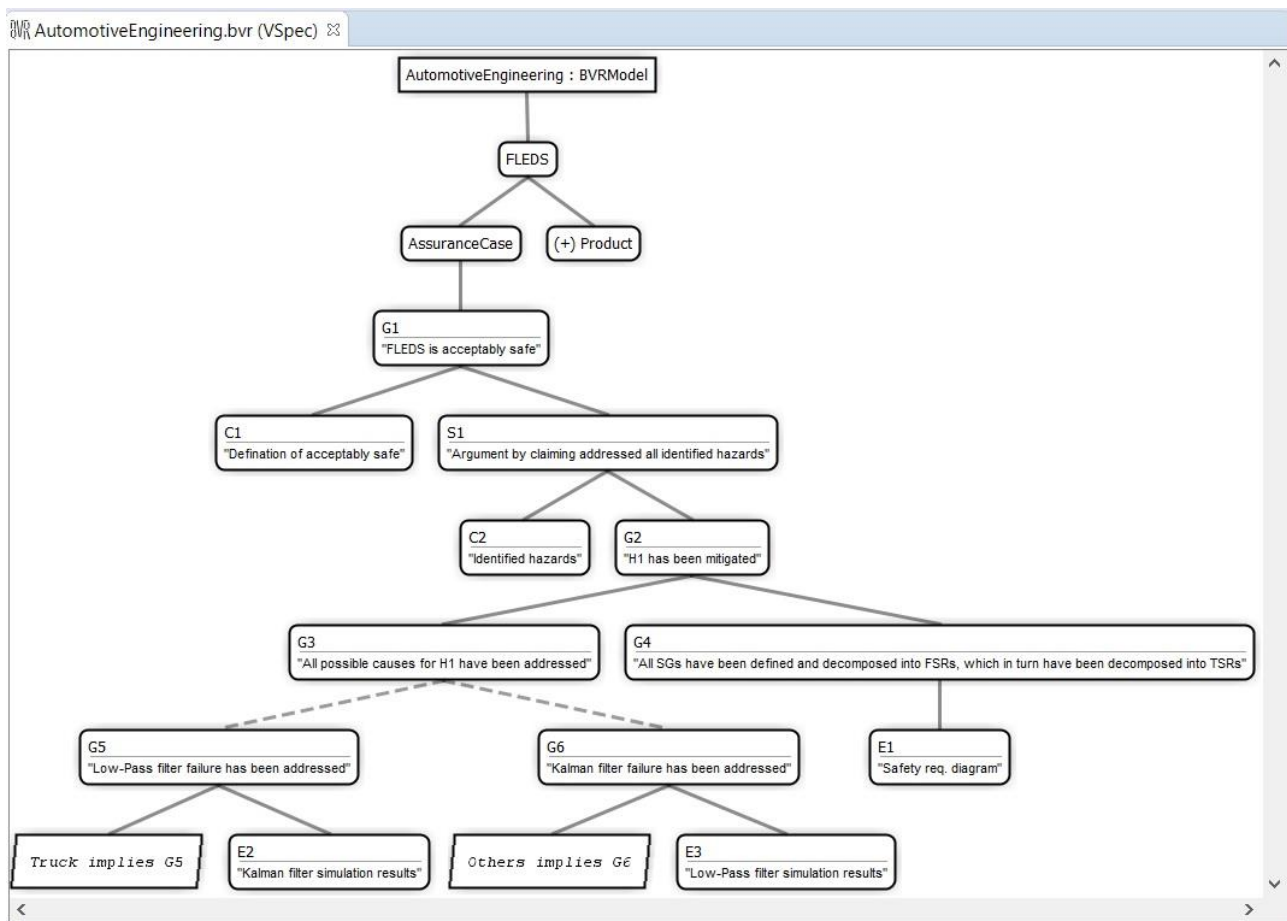
Figure 59, thus, represents one configuration of the family. This configuration (Base model) can be made vary via the designed integration of OpenCert and BVR Tool.



**Figure 59.** Argumentation for FLEDS in OpenCert

The VSpec model is shown in Figure 60. 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.



**Figure 60.** VSpec model regarding FLEDS

The resolution models, shown in Figure 61, represent the two possible argumentation configurations (one in case of Truck and the other in case of vehicle type different from Truck (Other)).

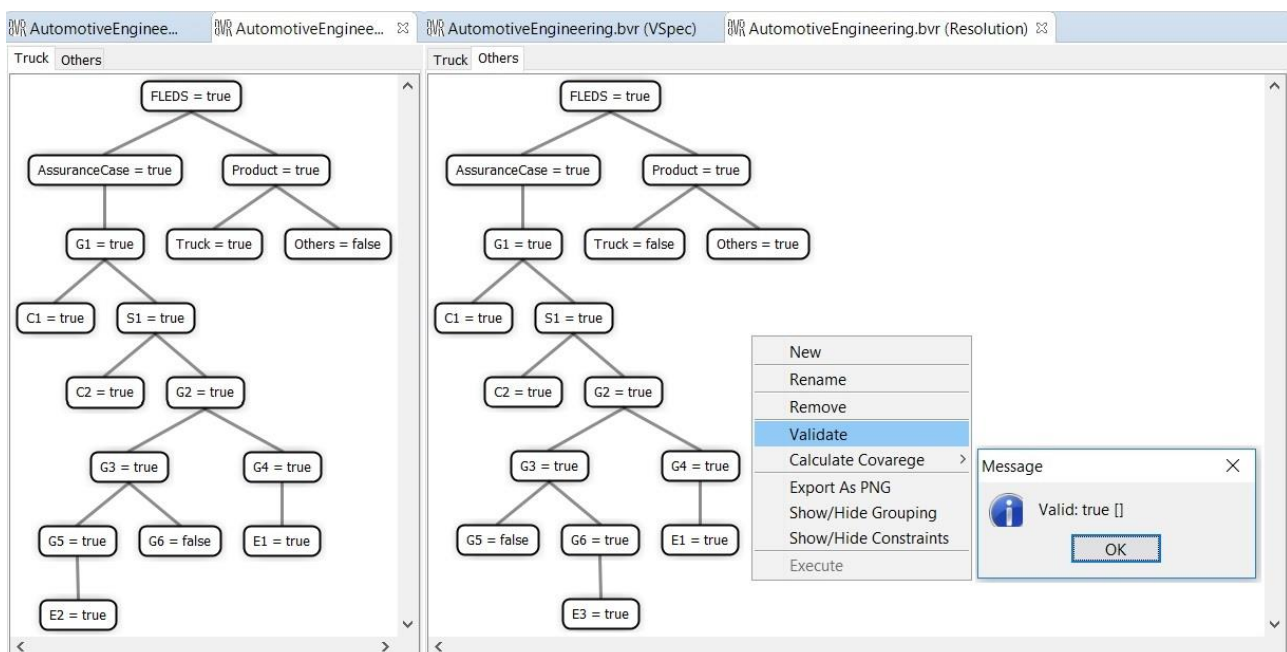
**Figure 61.** Resolution models regarding FLEDS

Figure 62 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 62, elements of a placement fragment are removed and elements of a replacement are injected. The placements are visualized in red. The re-configured model and diagram are updated in the assurance case editor of OpenCert. The new configuration, where G6 is present instead of G5, is given in Appendix B, Figure 131.

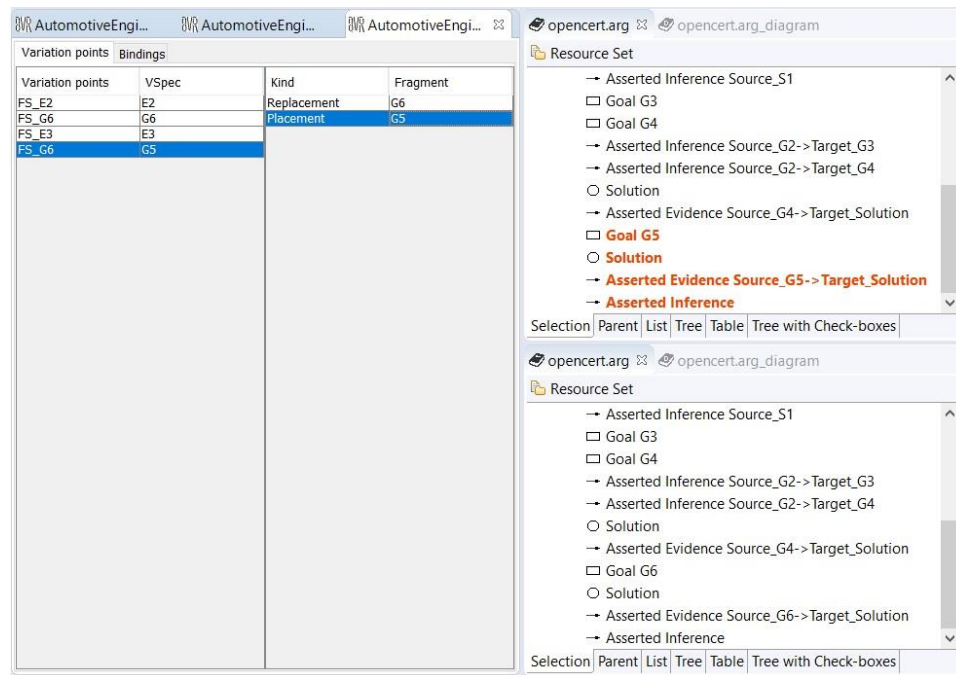


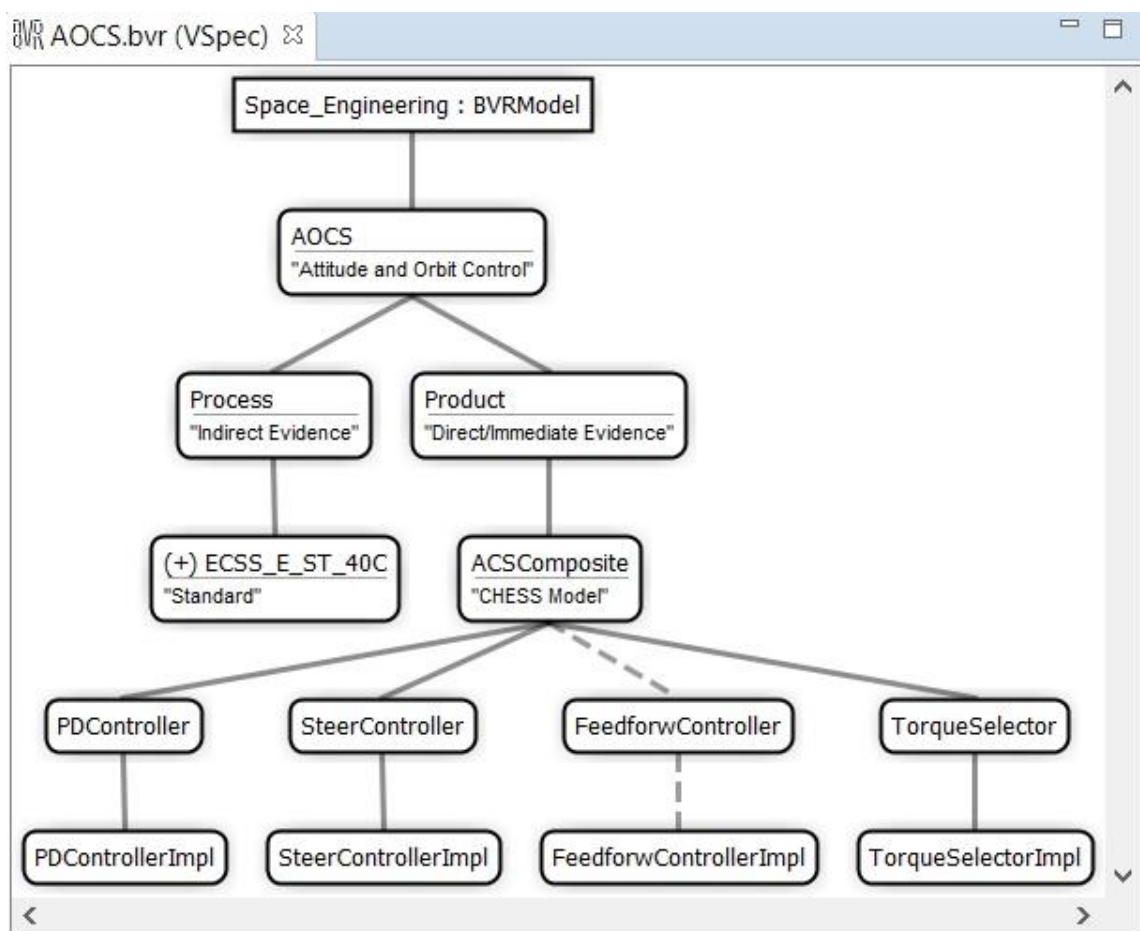
Figure 62. Realization model regarding FLEDs

### 5.3.5 Anti-Sisyphus: (3+1)-D Reuse and Impact Analysis via UMA, CHESSML, CACM, and BVR (\*)

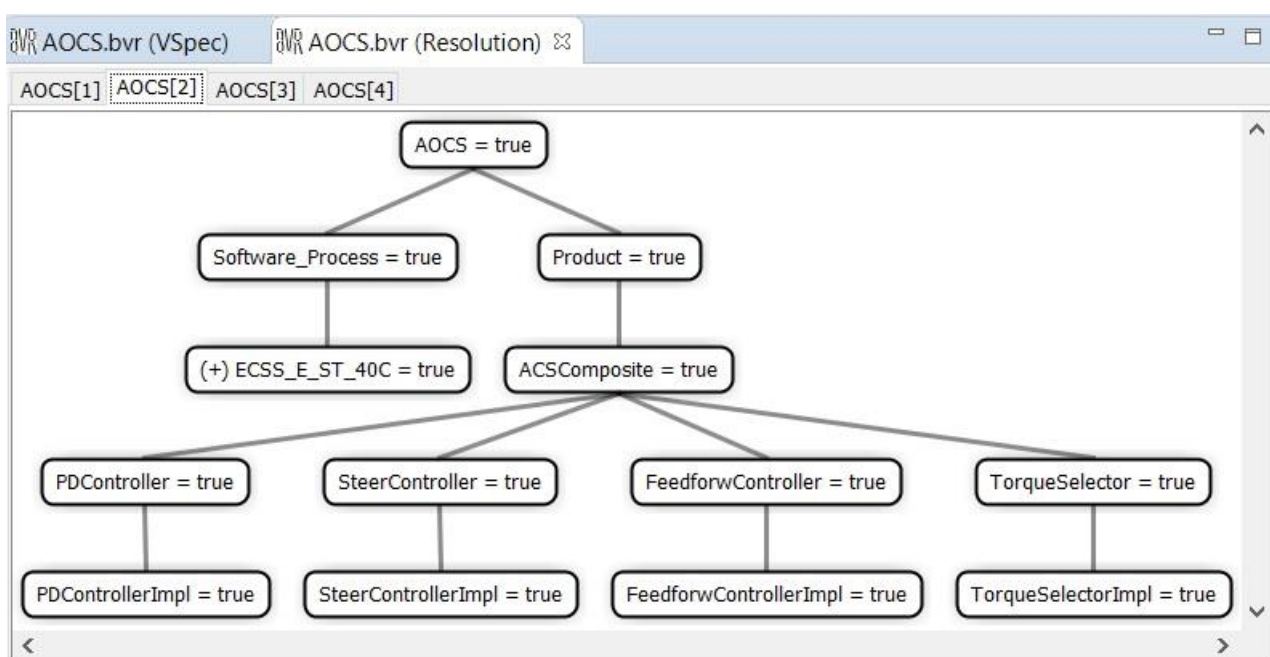
Previous subsections have presented how to systematize commonalities and variabilities within each quadrant of the 3-dimensional line, constituted of: process line, product line, and assurance case line. Besides these three dimensions, a fourth one can be considered and handled in a similar manner via the integration of EPF Composer and BVR Tool: the standard line, i.e., the set of standards belonging to the same family because of small changes from one version to the next (ISO 26262-2011 and ISO 26262 2018); because of cross-concern overlapping requirements (ISO 26262 (functional safety), AutomotiveSPICE (quality)); etc.

In this subsection, we further discuss how the different VSpec/Resolution/Realization models pertaining to the different dimensions could be linked to enable impact analysis and increased reduction of unnecessary repetitive actions.

If, for instance, a user with knowledge/expertise on both process and product information, wants to manage the variability regarding process and product, (s)he can create a single project, register the different editors and work for instance on a common VSpec (Figure 63) or Resolution (Figure 64) models, here minimized for space reasons but available in their expanded version in Appendix B. However, the execution of the changes regarding the different parts (process/product) can only be done sequentially, since BVR-tool executes the changes onto one Base model at the time.



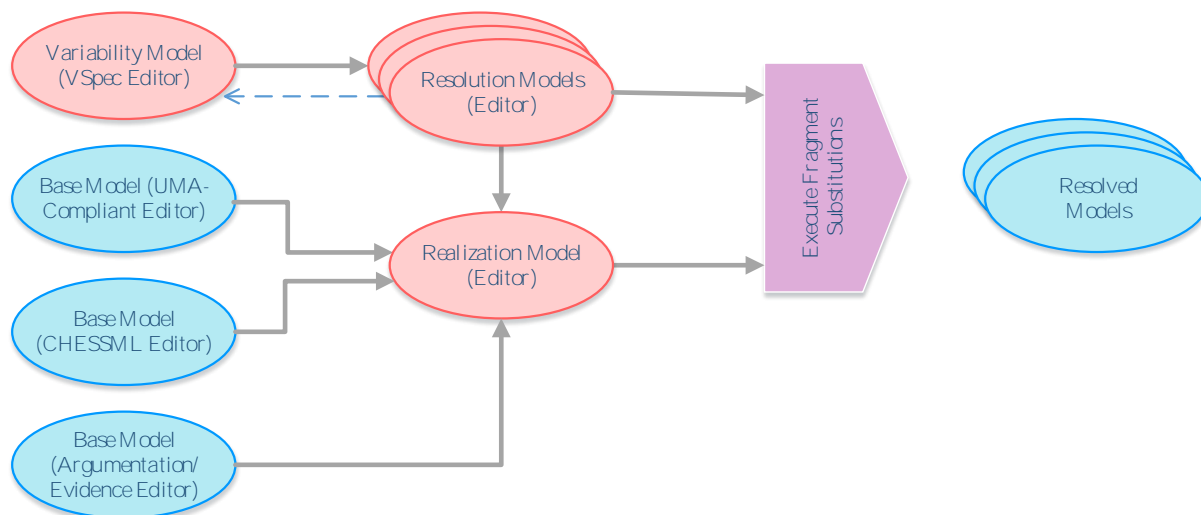
**Figure 63.** VSpec Model regarding ECSS-related SW Development Process and Attitude Control System (ACS)



**Figure 64.** Resolution Model regarding ECSS-related SW Development Process and Attitude Control System (ACS)

To enable process engineers working on process lines, separately from designers and assurance managers, an additional functionality should be at disposal, aimed at notifying related roles about the impact of the changes performed. More specifically, whenever a process engineer changes the configuration of the process in a way that can affect the designers and/or assurance engineers, impact analysis results should be performed.

This additional functionality permits to move from a mono Base-model-based BVR-Tool version to a multi Base-model-based version, where one editor is fully aware about the others involved.



**Figure 65.** Models interplay enabling management of process product and assurance case lines

The process, product, and assurance case variability might be specified in the combined or otherwise individual models, as shown in Figure 66. In the combined models, the individual branches might be taken into consideration for the process, product, and assurance case variability. In order to characterize the structure, a possibility is VType, in which unchanged structures might be defined; the nested elements cannot be retrieved at the resolution and realization levels. The constraints have been enforced over the model elements, for which their names are considered. It is therefore important to avoid duplicates; the occurrences could also be defined. Previously, the constraints were only supported for current model elements.

The idea with the individual models is separation of concerns, so that the process engineers, product designers and safety engineers work on their respective models. Therefore, the interactions between process, product, and argument models have to be supported; the logical operators such as implication, alternative, and negation might be used in the cross-cutting constraints. It is a convenient way to enforce the process, product, and assurance case relationships. There is also a need to support the occurrence specification between the variability models in a project.

The presence of the mentioned elements is first checked in the current model. If the elements are not detected, the search is extended to other models in a project. In case the elements are detected in another model, the dialogue window pops up to inform the existence in specific model. The user, however, needs to authenticate the enforcement of specific constraint or occurrence.

The support for error checking and validation of resolutions is incorporated for the combined models, but the individual process, product, and assurance case resolutions needs to be linked. The join option is incorporated for the process, product, and assurance case resolutions. In order to support the cross-dimension change propagation, the execution of variation fragments is simultaneously supported for UMA, CHESML and CACM compliant models. The results have been propagated back to the EPF Composer, CHES and OpenCert. Therefore, the cross-cutting constraints between the process, product, and assurance case dimensions via inter-model communication and cross-dimension change propagation are supported.

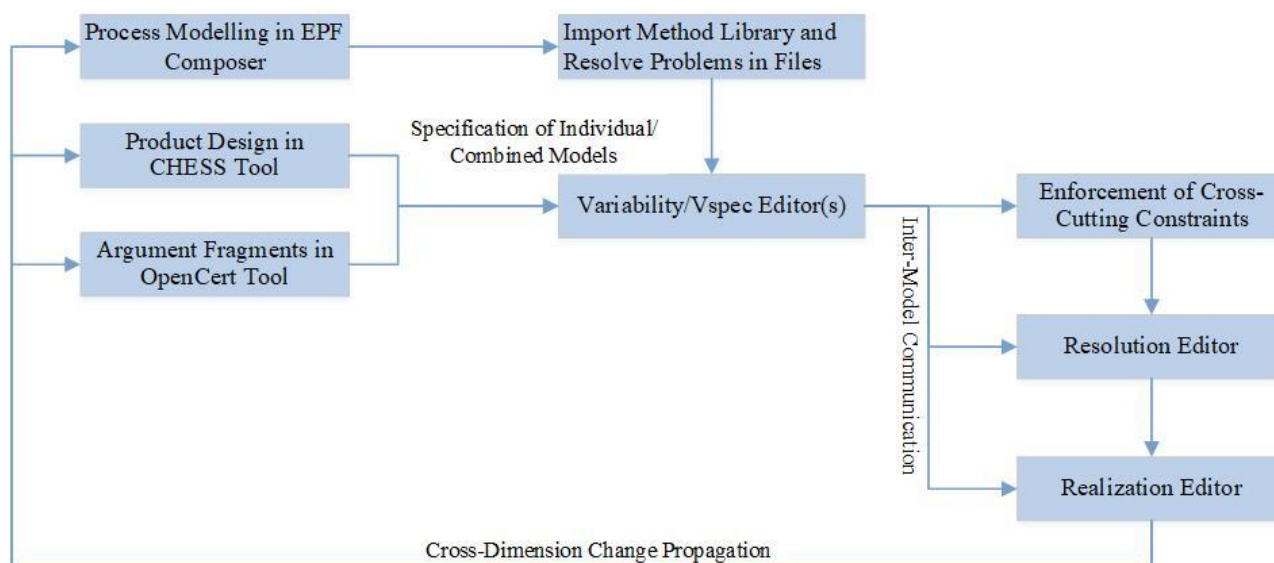


Figure 66. Impact analysis and change propagation in families/lines

## 5.4 Product-related reuse via MDE and meta-modelling: focus on analysis artefacts

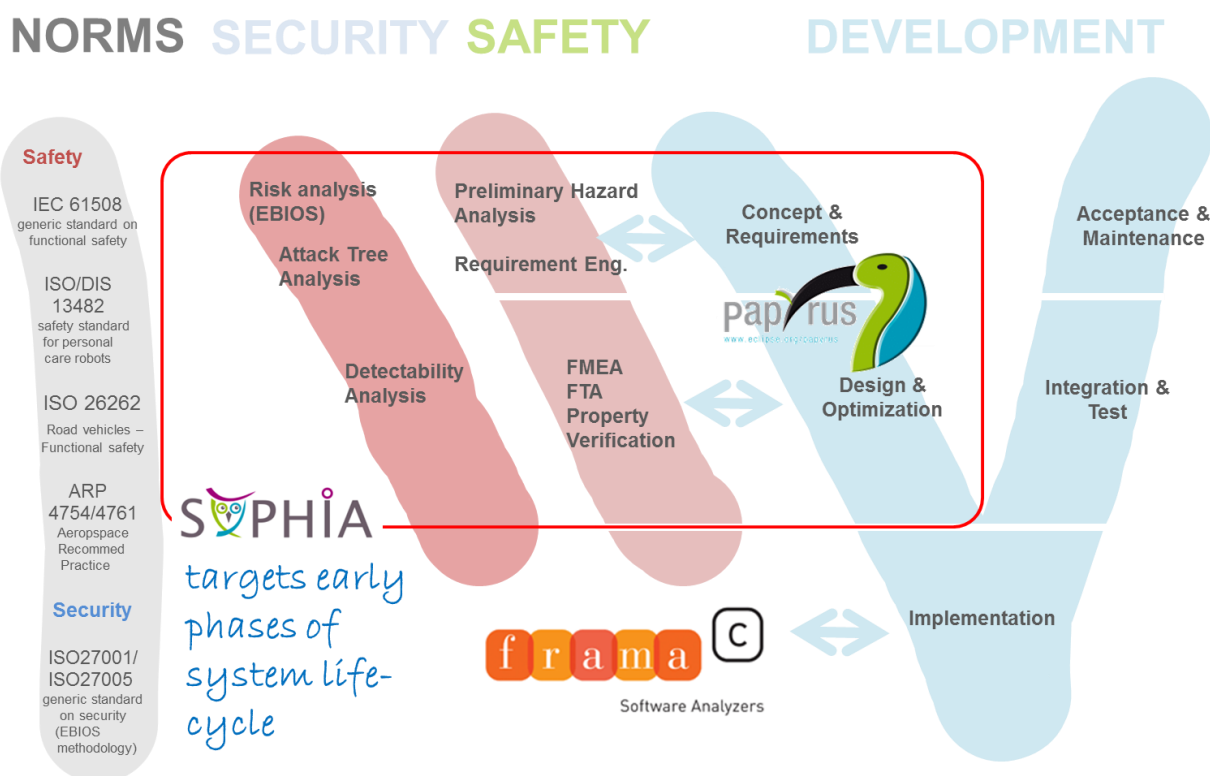
To enable product reuse, tools outside the ARTA (AMASS Reference Tool Architecture), e.g., tools for performing safety/security analysis, may take a different approach to face the challenge related to reuse of analysis-related artefacts. This approach may be based on:

- *Model Driven Engineering (MDE)*: MDE is proposed to achieve seamless product reuse from design. Indeed, design can take advantage of certain MDE approaches and languages like expressiveness, extensibility, usability, etc. in order to tackle the difficulties of product reuse w.r.t. to analysis notions, artefacts, methods, etc. In particular, the use of Model Based Systems Engineering (MBSE) is considered. More precisely, the commonalities and variabilities for safety and security analyses are also targeted. Indeed, the use of standardized non-proprietary languages like SysML and UML provides a basis upon which the different concepts related to regulation, methodologies, and knowledge bases can be modelled and coherently integrated and maintained.
- *Modelling and analysis support*: SysML and UML provide extension mechanisms that allow the specialization of existing stereotypes and the definition of new ones. Additional support is foreseen to improve automation of the following design process activities:
  - Standard/regulations capture and modelling (as processes)
  - Product/system modelling
  - Requirements management: specification, traceability, quality metrics and evaluation, validation & verification
  - Joint safety-security modelling and analysis
  - Knowledge bases import and export
- *Management of commonalities and variabilities*: a consistent management of commonalities and variabilities can be performed at meta-model level:
  - Identification of commonalities at product/system and analysis levels
  - Specialization of meta-models according to identified variabilities
  - Definition of associations to ensure elements traceability and consistency
- *Hostile context modelling*: The extension mechanisms of UML/SysML allow specializing existing diagrams in order to support modelling of Attack Trees, Attack-Defence Trees, Threat Scenarios, Misuse Scenarios, etc. Typical MDE frameworks can also be plugged to environments including an



attacker model that allows the validation of requirements and countermeasures efficacy. The use of referred frameworks, integrating attacker model(s) validation, can be shown in the context of the AMASS project [162].

As an instance of a framework exhibiting some of the features listed in previous items, we introduce the tool named Sophia. Sophia is a modular – for now, proprietary - tool developed and maintained by the CEA and is based upon Papyrus [108]. As shown in Figure 67, Sophia mainly supports system design from early phases of systems conception (left part of the V-cycle).



**Figure 67.** MDE-based seamless safety-security analysis targeting reusable products design

The framework is adequate to independently conduct safety and security analyses. The fundamental notions and methods, upon which the analyses rely, are mainly inherited from standards. As for safety-related standards, Sophia is based upon IEC 61508 (functional safety of E/E/EP systems), ISO/DIS 13482 (personal-care robots), ISO 26262 (safety of road vehicles), ARP 4754/4761 (safety of aerospace systems). As for security-related standards, Sophia is currently aligned with ISO 27001/27005 series. The Sophia modules allow conducting typical safety analyses like PHA, FMEA, and FTA. In addition, the security framework supports security risk, attack trees, and threats detection analyses. Such features allow the reuse of system models when both safety and security aspects need to be explored. However, further efforts are still necessary – and are currently in progress – in order to identify commonalities in safety and security analyses that can be integrated so as to ease product reuse.

## 5.5 Product-related reuse: focus on safety and security analysis artefacts (\*)

**Context.** The major interest of system attributes analysis tools, such as safety or security analysis tool, is to make the analysis easier. However, when a system model has progressed/modified, it can demand a huge amount of time to be analyzed again. In order to avoid this issue, the reuse features, such as Diff/Merge can compare and merge sets of model elements to prevent data loss and enforce model consistency during merge.



The Diff/Merge feature has been implemented in Safety Architect tool to merge two versions of the same model with *functional elements* (system components and component ports) in order to reuse *Safety or Security analysis artefacts*, such as failure modes, malicious events logical gates, and propagation links contained in the version already analyzed.

**Principle of the solution.** The solution implemented in Safety Architect is the EMF Diff/Merge<sup>12</sup>. The usage process is illustrated in Figure 68.

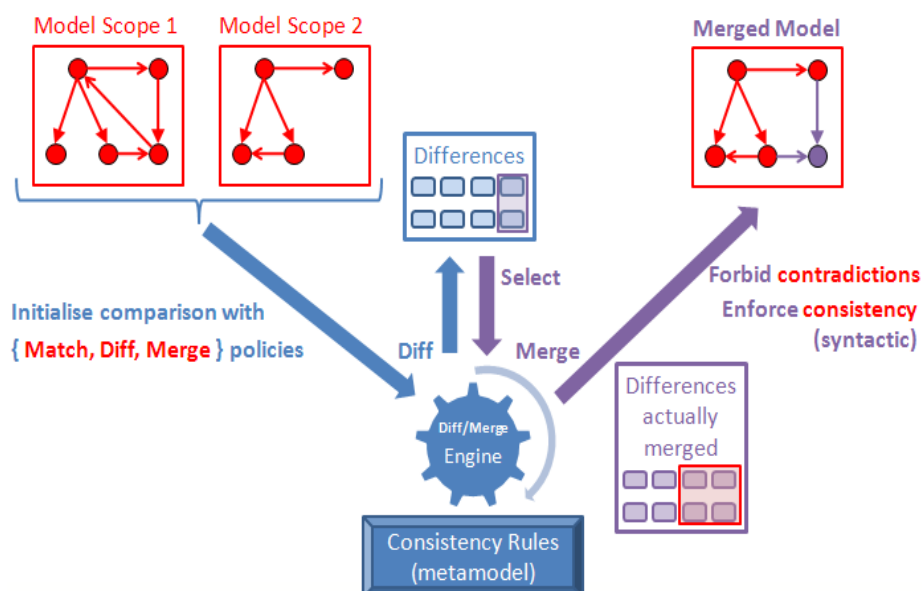


Figure 68. EMF Diff/Merge Principle

First, a comparison is created based on the models to compare. The differences between those models are computed according to given policies. Then, as long as differences remain, any subset of these differences can be selected for merging. Every time, predefined consistency rules and user-defined policies are used to compute the minimal superset of differences that must be merged to preserve consistency. The user may decide whether to confirm or cancel the merge of the whole set of differences.

**Diff/Merge implementation in Safety Architect.** The main idea of the implementation is to allow a user to enrich functional elements of a new version of Safety Architect model with dysfunctional elements added in an older version already analyzed into Safety Architect.

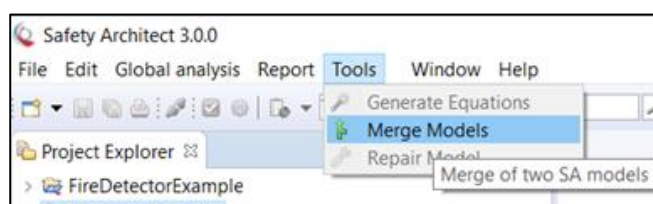


Figure 69. Diff/Merge implementation in Safety Architect

**Merge of two Safety Architect Models.** Thanks to this implementation (see Figure 69), the user can reference the name of the “reference model” (the Safety Architect model including dysfunctional elements), the name of the “target model” (the new version of Capella model) and the name of the “destination model” where the merge operations will be applied.

<sup>12</sup> [http://wiki.eclipse.org/EMF\\_DiffMerge](http://wiki.eclipse.org/EMF_DiffMerge)

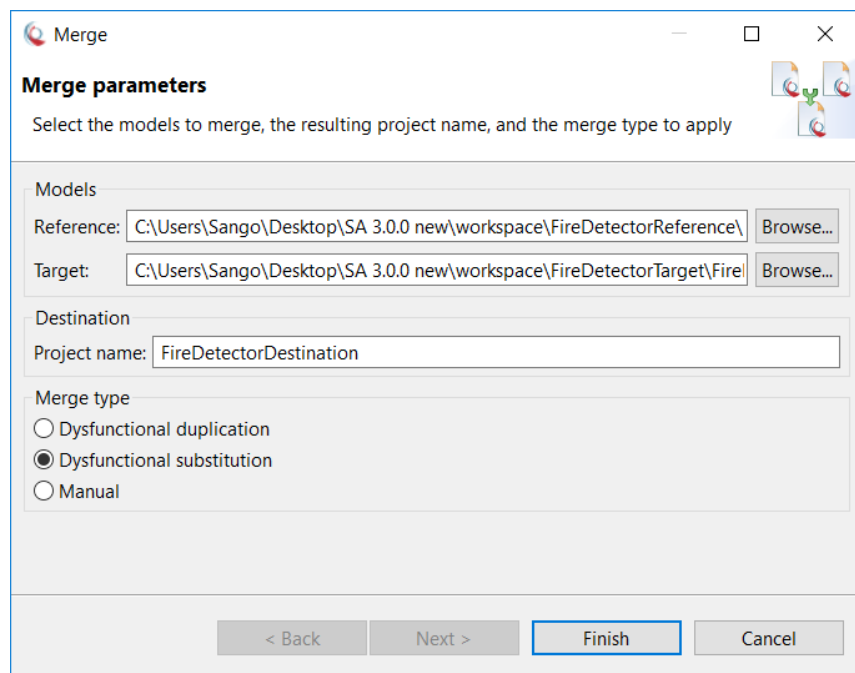


Figure 70. Merge of two Safety Architect models

Finally, the user can choose among several types of merging. For example, with *Dysfunctional substitution* (see Figure 70) all the dysfunctional analysis of the target model is substituted by the ones of the reference model. With the *Manual* merging, the user can see all the differences between the two models and can manually choose the ones to be applied, as illustrated below.

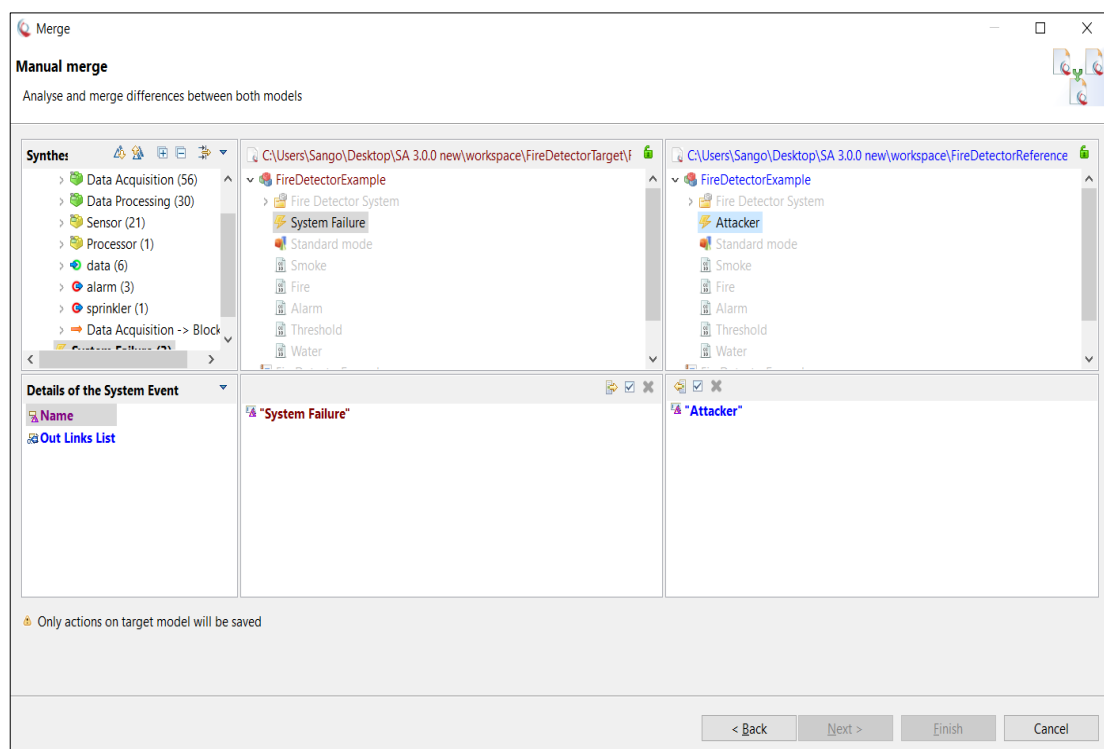
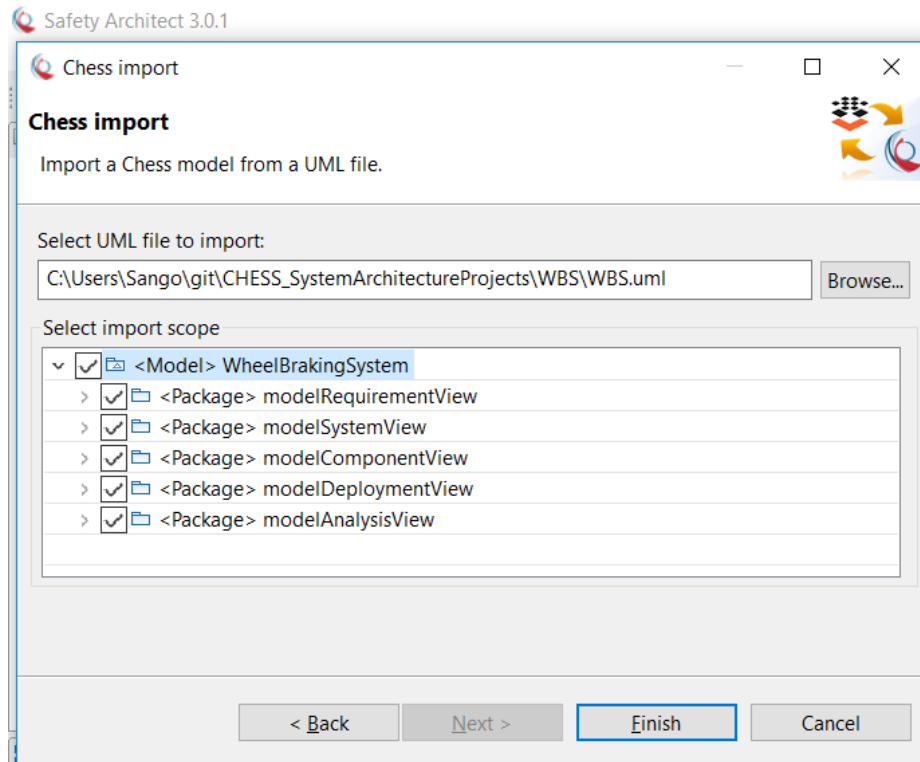


Figure 71. Differences between two models

**Application in AMASS platform.** AMASS platform includes among other the CHES tool, and an interface is developed between CHES and Safety Architect to import CHES model as illustrated in figure below.



**Figure 72.** Import from CHES to Safety Architect

The Diff/Merge reuse mechanism can be exploited during the **re-import** from AMASS platform - Chess tool to Safety Architect tool. Indeed, if there is at least one existing project (imported from the CHES) into the SA workspace, the user can apply the merge operation during the **re-import** of the same model. One of the scenarios can be:

- Import a CHES model to Safety Architect
- Perform Safety or Security Analysis in Safety Architect on the model
- When the same CHES model is modified, Re-Import the model by applying this Diff/Merger

## 5.6 Conceptual approach on product reuse (\*)

In chapter 4.2.11, an approach and aspects on “product reuse” were presented, aspects of component development (hardware or software) that can be built once and reused in other contexts. We emphasized the accompanying (safety) analyses and related documents which define the *context* in which the component has been developed. Conceptually, one can distinguish two cases:

1. Product development in *a system context*. In this case the environment of a component and its application context is fully known. During design, development, and verification all safety integrity requirements can be verified or validated.
2. *Safety Element out of Context* (SEoC) development. Designing a product needs to assume a certain system context, interfaces, environmental conditions, usage scenario, and application.

From a technical point of view these two cases are not much different, since a component has always dependencies to its surrounding system (and the system to the component, otherwise the component would not be required). The only difference is *when* exactly the final verification can take place, and more specifically, the time when the component can be configured to fit into a certain system context. For example, an



Electrically Assisted Power Steering (EPS) controller might be developed and analyzed with assumptions about a number of system characteristics:

- mechanical connectivity, such as steering column, mounting requirements, sensor placement, etc.
- electrical and communication interfaces, such as power plugs, CAN/FlexRay interfaces/messaging, wiring, etc.
- system parameters, such as vehicle size/weight, wheel characteristics, speed data/acceleration parameters, etc.
- Thermal and physical stresses, such as temperature, vibration, humidity, mechanical forces, etc.

If the development takes place in a system context, i.e. for a specific OEM and vehicle type, then most of the parameters are known during design/development time or, at the latest, during installation for certain calibrations (e.g. steering wheel return, “zero positioning”). If the development takes place out of context, all the interfaces and system context parameters need to be assumed and can only be validated (and to a certain extent verified) during application development. Similarly, the same applies if smaller components are developed - e.g. integrated circuits - or whole (sub-)systems, e.g. motor controller platform, ADAS functions such as lane keeping assistant, parking assistant, etc.<sup>13</sup>

In conclusion, at the core of any product reuse we see three important steps that need to be considered for safety critical reuse:

1. Definition of the dependencies of the component/subsystem that shall be reused. As outlined, this includes all related information possibly spread over several analyses, work products, etc.
2. Encapsulation of the component and its related data (“export”). This means to precisely determine the scope of information that needs to be exchanged with the component and extract it. All related data can be seen as the extended interface of a component ranging from safety requirements to loosely linked information, e.g. system parameters that impact the component, arguments in a safety case, assumptions or use, etc.
3. Embedding the component and the related data into the new context. This involves consistent adaptation and checking of all the encapsulated data and establishing the connections to the new context.

These three steps can be best manifested using a model-based and integrated approach to safety analysis and safety assurance. Our vision is that a safety tool or platform offer means to export parts of a safety project (merely reusable parts of a safety analysis) so that others can use the exported analysis in their project. We develop the concepts in the next section, along (re-)use examples outlined in section 4.2.11, namely FMEDA and FTA.

### 5.6.1 Reuse using a Model-based Integrated Safety Analysis Approach

As a pre-condition to our reuse approach is the definition of all related data using (object-oriented) *metamodels*. This has been conceptually described in previous deliverables for the AMASS platform (e.g. D3.3 [10] and others). Metamodels precisely define the structure of models (abstract syntax) including their relationship and, optionally, textual or graphical syntax as well as behavior [150].

The challenge is to define the dependencies of the component and its work products. Note that here we refer to development-related information. While safety contracts define the interface dependencies of the component in the system context (i.e. during runtime), we need to specify the relationships between work products such as safety concept (safety requirements and safety mechanisms), failure cause/effect relations, prevention/detection measures during design/testing/installation, and so on. In order to maximize the alignment of the safety aspects with the system model and to ease the dependency resolution, we propose to

---

<sup>13</sup> Of course, we simplify here some of the complexity of real-world system and component dependencies and influence of technology choices as well as implementation strategies (e.g. Software) for the sake of developing the concepts and design of our reuse approach



*integrate* the safety-related information to a large extent into the system models. Besides what has been the state of the art in safety tools such as Medini Analyze (see [151]), we would follow up on work and ideas of Kaiser et.al in [152] on “CFT2.0” and integrate the essential parts for the safety analysis into the component model in SysML. If the complete fault model is captured within the design model and can be *configured* and checked by means of semi-formal (system) context parameters, it’s ready for reuse and can be systematically transferred into a new application/product context with tool support. All remaining related information, such as safety requirements and finally the safety case, need to be transferred on a case by case basis.

The conceptual extensions for integration into a SysML (or similar component) model should be the following, besides the functional and physical interfaces of a component/sub-system:

- I. Intrinsic and extrinsic failure (modes) of components and its parts. The failures should be attached first of all to structural elements such as blocks, ports, connections, signals, but also to certain behavioral concepts, e.g. state transitions or actions/activities.
- II. Failure rates that can be parameterized by a mission profile and distributed over the failures and/or internal component structure. A mission profile should consist of all data that is used for the safety and reliability characteristics (e.g. temperature cycles, expected system lifetime)
- III. Safety mechanism (SM) built into the design. Thereby, safety mechanisms can exist as integral part of a component (e.g. ECC/EDC in a memory component) or external to a component, e.g. allocated software or surrounding hardware (e.g. watchdog).
- IV. Diagnostic coverage (DC) provided by a safety mechanism and its relation to which kind of failures are covered - potentially with refined values for different activation states of the component (e.g. startup, operation, standby, degradation modes, etc.)
- V. Failure relations:
  - A. Fault propagation should be modelled within a component and to/from its interfaces as well as how failures can be detected by means of SMs.
  - B. Failure effect hierarchy, i.e. across abstraction levels of the design up to potential hazards/failure conditions at system level
  - C. Activation of safety mechanisms and relations to the fault propagationNote that we see different possibilities to express these relations, CFT being one option, extended failure net models being another.
- VI. Safety contract that is implemented by the component and/or supported at its interface. This should cover most of the semantics part for runtime safety
- VII. Design Controls for Prevention and Detection, generally referred to as safety measures e.g. as identified during an FMEA that address systematic failure causes.
- VIII. Requirements (especially safety) for the component addressing implementation, development, production, maintenance, etc. for the reused component. This could be anything that can’t be formalized within a design model. Assumptions of use, safety goals/safety objectives, reliability and availability requirements are subsumed under this point.

All this needs to be modelled and is captured specifically in safety analysis such as FMEDA and FTA which are important to be “interpreted” but also customized in a new system context.

## 5.6.2 Implementation Approach and Use Cases

Regarding the state of the art, the exchange of safety results, of hardware metrics of reusable assets for composed safety cases is not tool-supported nowadays, due to missing standard formats for safety analysis, but also sometimes due to IP problems or legal issues. We will investigate in the next phase on the modelling of the aspects listed in the previous section along an exemplified reuse of a configurable component at hardware level, namely an IC.



Our vision starts basically with semiconductors, onto which (according to the upcoming ISO 26262 standard v2) hardware safety analysis has to be performed. Such analysis is in turn required at integration level by their customers. The crucial points are as follows:

- (Re-)Use does not necessarily mean “import”, it is more important that customers can see the analysis result but also configure (i.e. adapt) the semiconductors to their specific system environment, specifically important for SEooC, i.e. usage of a certain IC in an ICU, software safety mechanisms, etc.
- The vision is that the export allows the semiconductor to hide its IP, its detailed analysis methods, the real formulas behind quantitative results, but still give the customer enough flexibility to adapt the usage of the chip, IC or whatever hardware in their design. But we exclude any further legal objectives since they can hardly be covered in the remaining period of the AMASS project.

For example, the semiconductor supplier may define a set of safety mechanisms that CAN be applied and that are OPTIONALLY available in the chip, the customer may do so and switch on/off certain mechanisms without the need to know how the switching on/off impacts the hardware metrics while still the calculation of the overall Hardware Architectural metrics (SPF/LF Metrics) of part 5 of ISO 26262 for the system is affected by the configurations.

## 5.7 Model Based Testing for exploring the benefits of re-use of development cycles (\*)

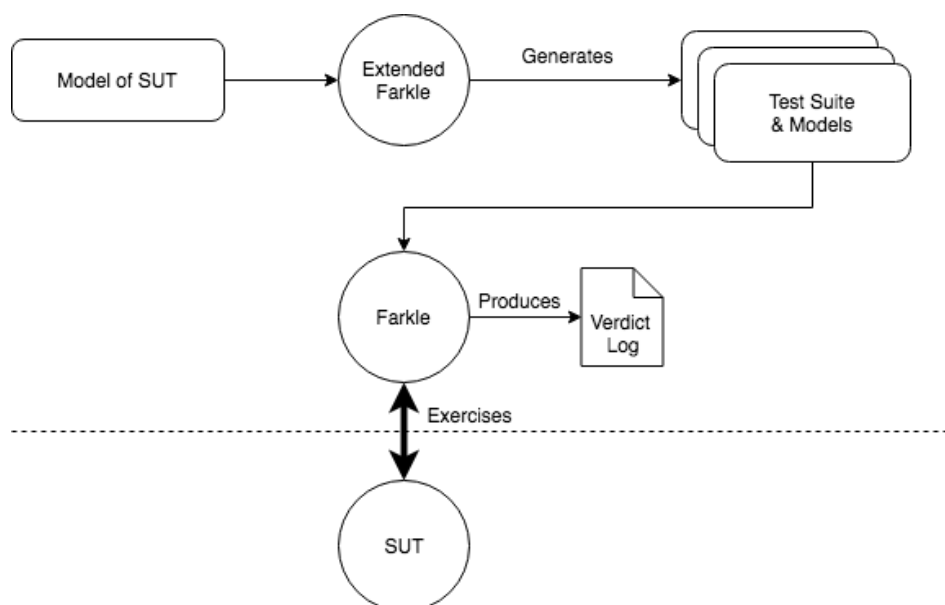
Model based testing is a relatively new process in software engineering. Current praxis in the industry is to handcraft tests to achieve a desired level of coverage. The coverage is calculated as described in the section on coverage. The problem is that handcrafting test to achieve a certain coverage can be time consuming and thus costly to write. Furthermore, achieving a certain coverage does not necessarily mean that the software is free of errors, having 70%-line coverage speaks nothing to the quality of the tests [144][145].

While model-based testing has not yet penetrated the industry sufficiently, it has been subject of research, and plenty of tools are now available to help developers. The goal of model-based testing is to generate tests automatically based on a model of the System Under test (SUT). While the number of generated tests can be infinite, developers can use rules to limit the generated tests. The greatest boon is gained from coupling automatic test generation with automatic testing, i.e. some type of framework which can run the generated tests automatically. The main problem with generating tests is that the model needs to be correct, if the developer writes an incorrect model the generated tests are most likely wrong. Despite this, model-based testing shows great potential in reducing test-time and simplifying test development while improving test efficiency, for instance, when developing a SUT, manual tests would need to be rewritten while the model-based method only requires an update to the model which then generates tests.

### 5.7.1 Automated Model Based Testing (\*)

Extended Farkle is a tool which provides test generation combined with test execution. In order to achieve this, Farkle is built upon third party tools and tools developed by Alten. It has been fine-tuned over many years, increasingly adding functionality to the tool. Using a Model of the system under test and the safety requirements of the software to be tested, also known as the SUT (System Under Test), extended Farkle generates a test suite along with "views" of the code, these views are for instance UML models. A Python interpreter executes the test suite on the SUT and outputs the results into a verdict log. This verdict log can be used as a fingerprint of sorts, when the SUT is rebuilt the same test suite can be executed and the differences of the results, the verdict delta, can be used to determine if the functionality of the SUT has been changed by the underlying changes to the code.





**Figure 73.** Overview of the Farkle implementation of model-based testing

## 5.8 Approach on impact analysis and delta analysis based on data indices using Elasticsearch (\*)

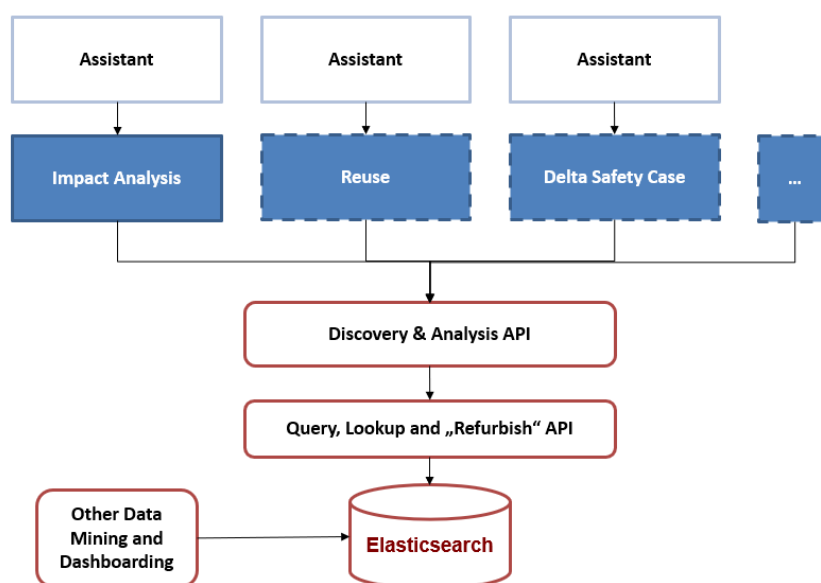
“Impact analysis”, “Assisted reuse”, “Delta safety case” and even the application of a “Safety Element out of Context”, all these aspects have in common that they need to search and query a huge set of data to provide answers. Criteria and problem scope are similar, as this example from practice shows:

A truck vendor has successfully accessed an E/E component for its standard short-haul truck, based on several *assumptions*, for example the overall truck length and the height of the driver cabin. The vendor plans to develop a special custom variant of the truck with a shorter length and wants to judge the *impact* on the safety case. In another scenario, the vendor needs a decision whether or not he can *reuse* the same E/E system in a garbage truck which typically has different driver cabin formats. Different questions in detail, but similar in general.

In a model-based environment with different underlying technologies and availability of data, it is as with other data mining approaches, key data are best indexed into separate optimized databases that are designed for large scaled and fast searches and queries. In WP5, the usage of Elasticsearch – an open source index and search database<sup>14</sup> – and Kibana – a graphical visualization and dashboard authoring system based on Elasticsearch<sup>15</sup> has been investigated for data mining and visualization with success. Consequently, building rich tool features, such as “Reuse Assistants” or any kind of “Impact Analysis” based on the same technology, is an option at hand.

<sup>14</sup> <https://www.elastic.co/products/elasticsearch>

<sup>15</sup> <https://www.elastic.co/products/kibana>



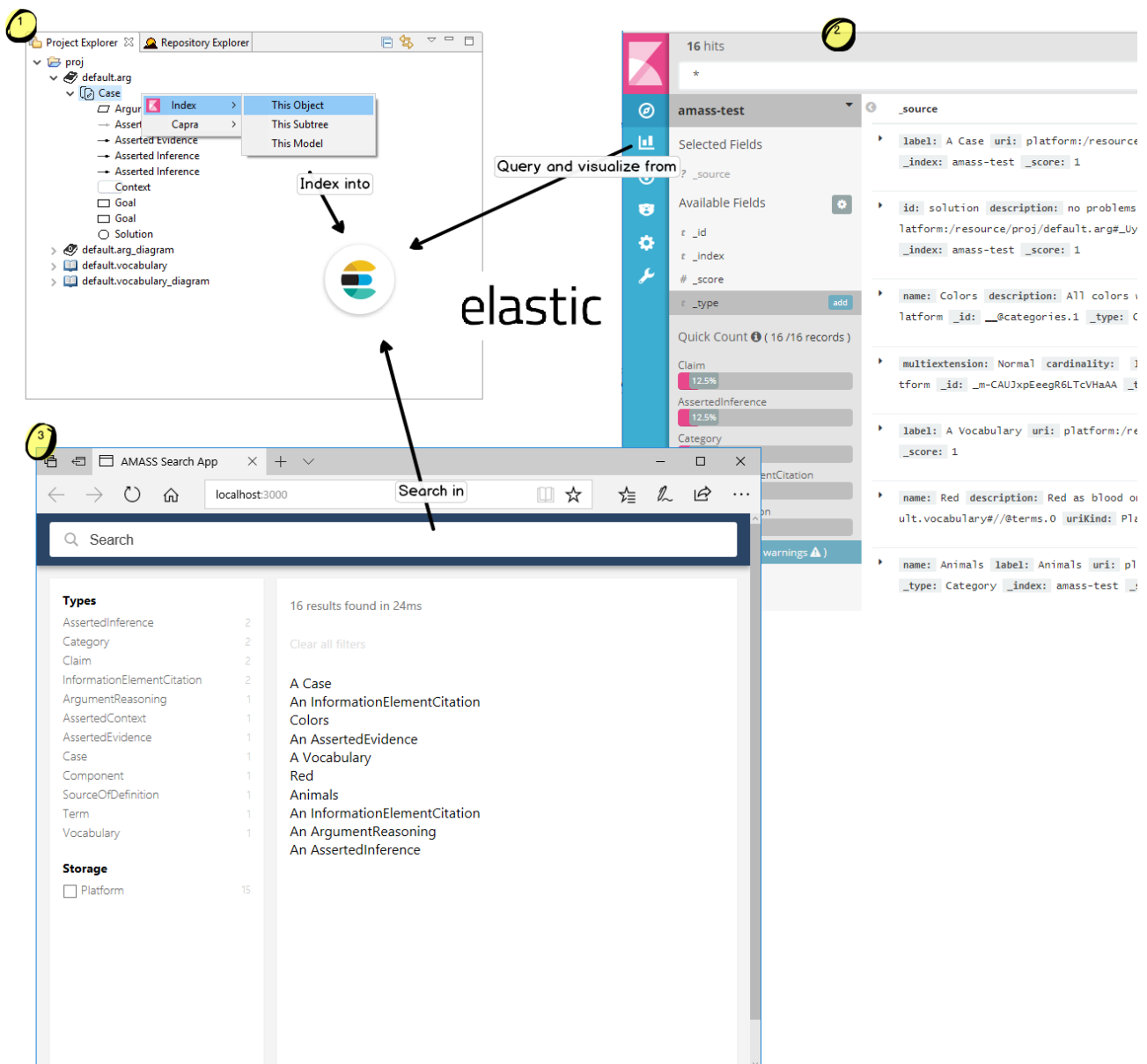
**Figure 74.** Elasticsearch overview

Figure 74 envisions two additional layers on top of the standard Elasticsearch API. The first (lower) one just “enriches” the basic Elasticsearch API by offering ways to map simple semantic queries to more complex Elasticsearch search requests and vice versa, i.e. combine and “refurbish” search results to pass simpler and less results to the caller.

The higher-level API is the one that offers discovery and analysis functions to higher level features (which are not further analyzed here). That API uses the lower level API but itself abstracts completely from Elasticsearch if possible. It can be treated as a reference implementation of such an API on Elasticsearch instead.

### 5.8.1 Achievements in P1

In Prototype P1, an Elasticsearch based index and search component has been developed. The heart of the component is a generic index feature for EMF based models. Based on the EMF Ecore metamodels, the indexer can index arbitrary an EMF object and structures into Elasticsearch by analyzing their metamodels, thus in a complete reflective way. The indexing is limited to be manual and actively executed in the prototype working on a set of selected elements, whereas it is independent on where these objects are located, whether they reside in the file system or on a CDO server does not matter. All indexed data can be directly searched and visualized using the Kibana dashboard software that accompanies Elasticsearch.



**Figure 75.** Interaction between P1 indexing tool, search app and Kibana dashboard

In addition to that, a sample web-based search application is available that - google like - allows a more user friendly and focused search than the Kibana dashboard, which is rather a data mining tool than a search app. The search app offers a set of exemplary filters to at least demonstrate the possibilities of semantic filters in the app. Having a separate filter, "Storage" was added simply to show that also aggregated metadata can be indexed and later used to improve the search result if necessary.

## 5.8.2 Challenges

There are several challenges in the envisioned approach but also in the results and lessons learned from the component developed in WP5 that was mentioned above.

The generic EMF indexer showed that it is in principle possible to develop a generic and reflective indexing for EMF models. However, metamodels are typically not "rich" enough to support the indexing process. So is it for example possible to annotate, at meta classes, which attribute shall be used as the "id" of an object, or whether certain attributes are volatile or transient, all information that may be used at indexing time. And there are a few further possible annotations to enrich the Ecore with more semantics. Typically, this information is not well maintained in all Ecore models.

In addition, the indexer just works on “metaclass level” so on a granularity that grants each object the same way. That means that the indexer maps objects to Elasticsearch documents one to one. Each object is stored in Elasticsearch as a single document, more coarse grain structures are not considered because they cannot be derived from the metamodel directly.

The key challenge however is to be able to map (or translate) the semantics of “an impact” - which is what upstream functions as impact analysis or reuse assistance are all about - to “search and match” - which is what Elasticsearch offers. So, the capability to start a query to Elasticsearch in a way that the result of the query not just gives a “text-based match”.

Last but not least, Elasticsearch itself introduces a number of challenges. The advantages - powerful search engine and match logic - is countered by a number of disadvantages. First of all, most Elasticsearch documents are “flat” documents, at least that is the preferred and advised way to store data in Elasticsearch to gain the best performance. Consequently, there are no relationships, data of related elements have to be “denormalized” at index time. Figure 76 gives an idea on how relationships can be denormalized into documents.

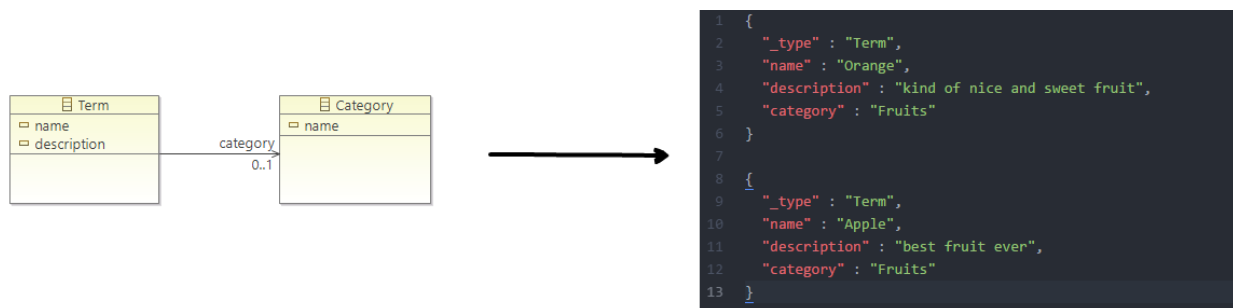


Figure 76. Relationships denormalization

In order to make reliable answers on impact requests, the data of the component must be enriched by a set of attributes that can be searched, compared and joined to make a decision on impact or reuse capabilities. A “reuse” answer here is treated as a “match” of a reuse request on the existing data. Referring to the above truck vendor example, it must be possible to express a query asking for the impact on changing the truck length or having or not having a cabin. It also must be possible to know the relationships of the mentioned properties (truck length, cabin presence) to safety critical functions in the component, for example as assumptions. So already when indexing any safety case data into Elasticsearch, this information must be resolved and added to the stored “safety case” document. The challenge is the way to express an assumption or a property of the safety case, a general concept is required to express this kind of attributes and in contrast to WP3 this requires a solution to be expressed in simple textual form.

### 5.8.3 Way forward / next step

The approach is promising although a number of challenges are already identified. There are basically three major areas that can be approached next and independent of each other.

Firstly, the way how data are extracted from models and indexed into Elasticsearch must be improved. A pure reflective approach is not sufficient. For specific types other means have to be found to introduce special navigation of data structures or collection of data from referred objects and models. Remember that “denormalization” is key here, so all safety case related attributes and assumptions have to be indexed in a flat manner. And they have to be somehow marked as special attributes to be later used during match and impact requests. Technically, certain extension points can be introduced for example to drive the indexing and navigation for dedicated meta class, as process or argumentation models for example.

A second important issue is the mentioned way to bridge the gap between the typical Elasticsearch text-based match approach to a rather “impact”-centric approach. Elasticsearch uses a defined built-in set of so called “Analyzers”, basically logic that basically is tokenizing a block of text into individual terms suitable for use in



an inverted index. To underpin the problem to cope with, just an example. An assumption may be given as plain text like the following:

```
> "The length of the truck is 18m"
```

The standard analyzer is the default analyzer that Elasticsearch uses. It is the best general choice for any text, in any language. It splits the text on word boundaries and removes most punctuation. Finally, it lowercases all terms. That would produce:

```
> "the, length, of, the, truck, is, 18m"
```

This is good for search performance but not sufficient for what we want to achieve. There are more default built-in analyzers (e.g. for different languages) but they do not change the general picture on how Elasticsearch works. By default, Elasticsearch configures text fields as "full-text" and applies the mentioned standard set of analyzers. Fortunately, custom analyzers and field types can be configured in a custom mapping - for example to index the exact value passed in, without any analysis, such as a string user ID or an internal status field or tag. The same mechanisms could be used to configure own analyzers to deal with the mentioned special attributes, such as assumptions or contractual attributes, later required to help in the impact analysis request.

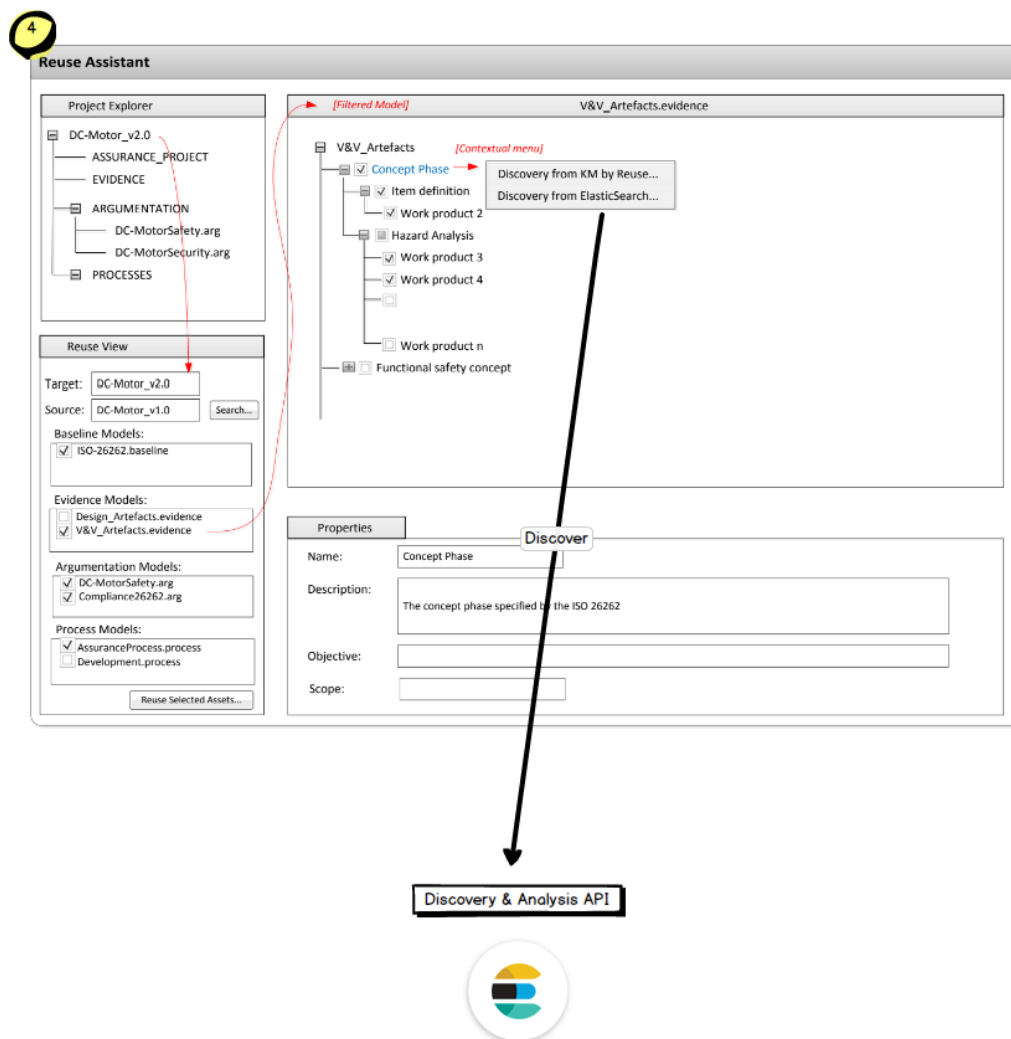
Another point, not yet considered, is the fact that in an impact analysis often a quantitative result is of interest, so not only that there is an impact but also how high the impact is. To achieve such a quantitative result, somehow numbers have to be associated with a document and/or its attributes so that a "match" is not only qualitative but also quantitative. Fortunately, Elasticsearch comes with an internal scoring and relevance system anyway (score, as depicted in Figure 77, is returned as metadata of the search result).

```
{
  "_index" :      "us",
  "_type" :      "tweet",
  "_id" :        "12",
  "_score" :      0.076713204,
  "_source" :     { ... trimmed ... },
```

**Figure 77.** Score information as meta-data

There is a lot of theory behind, but generally speaking: there should be a way to utilize that scoring system to quantify search results.

Last but not least, the high-level API that sits atop the bare bone Elasticsearch query and search API, and that can be used by any of the mentioned impact (at all) features and any assistance above, needs discussion with the use case owners. The API needs to be driven forward to requirements and needs of the feature themselves while considering the capabilities of (1) the approach and (2) the Elasticsearch capabilities and limitations.



**Figure 78.** Utilization of Elasticsearch based APIs

At the end the approach to serve multiple different use cases, from impact analysis to reuse via a single API and mapped to Elasticsearch, must be proven right.

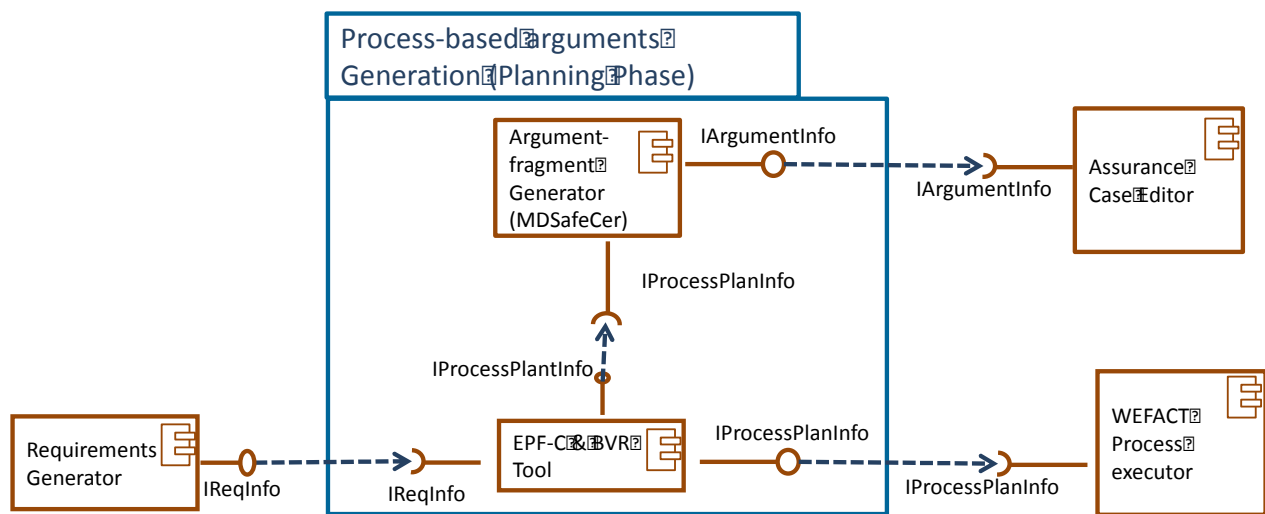
## 5.9 Automatic generation of process-based arguments

This section provides the design solution for the automatic generation of process-based arguments within AMASS. The solution embraces both phases of the *certification liaison process* (which is explicitly defined within DO-178C and implicitly in place in all certification/qualification frameworks): the planning (see Section 5.9.1) and the execution (see Section 5.9.2) phase.

### 5.9.1 Generating Process-based Argumentation Representing Plans (\*)

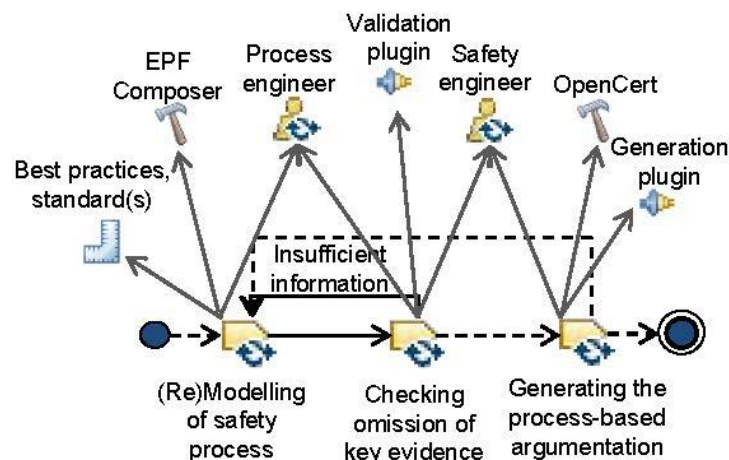
As depicted in Figure 79, during the planning phase, a specific process plan is derived from the family of processes (managed by EPF-C & BVR Tool), then an argument is automatically generated by following MDSafeCer [29] principles and visualized via the Assurance Case Editor.





**Figure 79.** Process-based arguments generation (Planning Phase)

Process-based argumentations argue that a safety-critical system has been developed in compliance with the development process defined in the standards and provide the evidence for certification of compliance. However, the process-based argumentations cannot ensure that the evidences are sufficient to support the claim. In particular, inappropriate, incomplete or inherently faulty reasoning about the evidence introduces the defects in safety argumentations, namely called fallacies. These fallacies could lead to overconfidence in a system and tolerate certain faults which in turn contribute to safety-related failures of the system. This risk also affects process-based argumentation. For example, a process-based argumentation, supported by the evidence of personnel competence in performing the model checking task, is weakened in detecting all faults in design because underlying proof attributable to a lack of training in formal methods. The undetected design faults during the development process might lead to the failure of a safety system when it is deployed. Therefore, it is necessary to prevent or detect fallacies in the process-based argumentations. In this context, a plugin will be developed that validates the process models, and prevents the occurrence of fallacy, specifically, *omission of key evidence* in process-based arguments. If fallacies are detected in the process models, the approach develops the recommendations to resolve them; afterwards the process and/or safety engineers modify the process models based on the provided recommendations. Finally, the safety arguments from the modified process model will be generated using the Generation plugin [174]. The overall workflow of the method is shown in Figure 80, specifically the solid lines show the extended step of MDSafeCer.



**Figure 80.** Overview of the proposed method



### 5.9.1.1 Types of fallacy (\*)

As discussed above, an argumentation fallacy is a mistake or flaw in the reasoning of an argument. In safety arguments, fallacies exist in different forms. Greenwell et al. presented a taxonomy of common fallacies in safety arguments and organized them into three categories namely, relevance, acceptability and sufficiency fallacies [148], described as follows:

- *Relevance fallacies* add no value to an argument and provide irrelevant evidence. The existence of a relevance fallacy in an argument cannot contribute to a failure; rather, these fallacies might mislead/distract the developer or reviewer into accepting an insufficient argument, which, in turn, may contribute to a system failure.
- *Acceptability fallacies* are those in which an argument provides unacceptable, contradictory or inconsistent evidence to support the claims, for example, an argument contains the evidence that is only the restatement of the claim.
- *Sufficiency fallacies* are those in which arguments can fail to provide sufficient evidence to support the claims, either providing little or no evidence, biased or weak evidence, or omitting crucial types of evidence. Within the context of AMASS the sufficiency fallacies have been considered. More specifically, *omission of key evidence* in which arguments fail to provide key evidence that is crucial to support the claim. Omission of key evidence fallacies, within the context of process argumentation, are the flaws or defects in which arguments can fail to provide sufficient evidence (e.g., staff competency) to support the process claim (e.g., claim about designer who is responsible for the design task, which deals with the production of design-related work products).

### 5.9.1.2 Modelling of Safety Processes (\*)

As shown in Figure 80, the first step involves modelling of a safety process in EPF-C according to the best practices as well as according to the standard(s). There are two possible ways of modelling requirements in EPF Composer. First, the requirements and associated process life-cycle can be modelled by following the IBM approach. However, EPF Composer does not support the definition of a user-defined type. Therefore, the guidance type Practice has been customized with an icon and variability relationships, as shown in Figure 81. The associated process elements (e.g., tasks, work products and roles) are modelled under the Method Content package and development life-cycle under Processes in the process-lifecycle plugin. The detailed guidelines are provided in the user manual.

Second, ECSS-E-ST-40C standard requirements can automatically be imported into the EPF Composer. For this, ECSS Applicability Requirement Matrix (EARM)—ECSS in MS Excel format (i.e., EARM\\_ECSS\\_exportDOORS-v0.5Statistics.xlsx has been parsed and the set of requirements has been filtered. The instructions for converting excel file to xml format compatible with EPF Composer has been described in AMASS D1.4 [147].

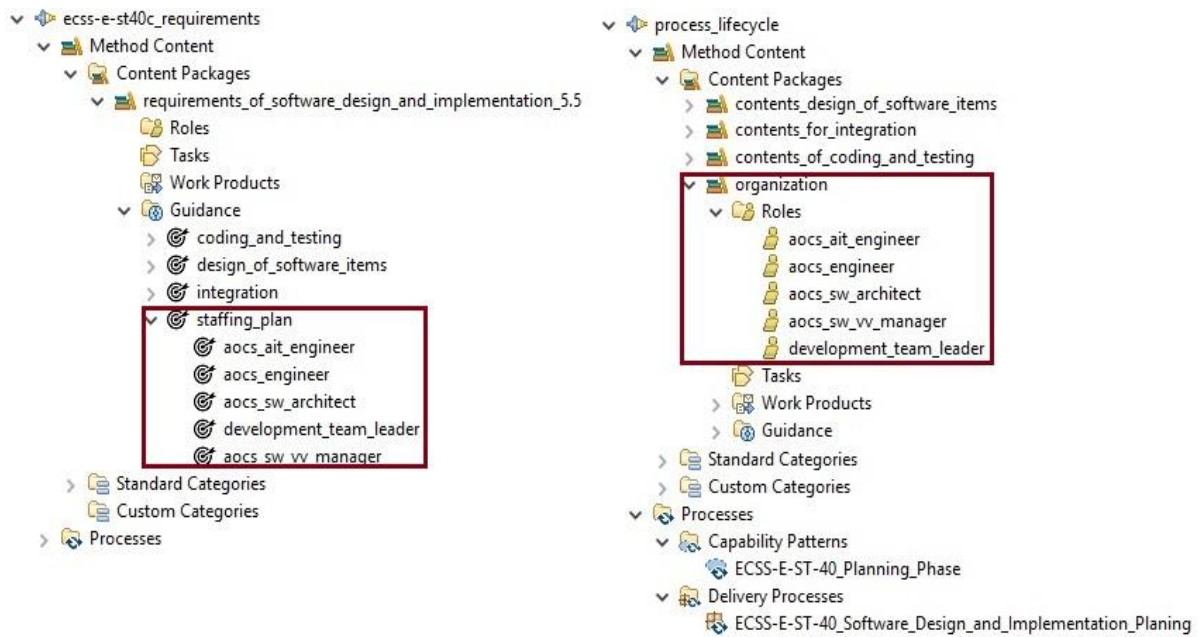


Figure 81. Requirements and Process modelling in EPF Composer

### 5.9.1.3 Detecting Fallacies in Process Models (\*)

This subsection explains the algorithmic design solution for detecting fallacies, more specifically, omission of key evidence fallacies, in process models. The approach validates whether the safety process contains sufficient information corresponding to the key evidence for supporting the specific requirement. Algorithm starts by searching the process model (i.e., top-level element) and considers the (decomposed) linked elements such as phases, activities, tasks and so on. In particular, it follows the *Work Breakdown Structure*, as shown in Algorithm 1. The function *getTopLevelElements()* returns a set of top-level elements of the process. Given a certain element  $e$ , its linked element  $le$  can be achieved by using the function *getLinkedElements()*. Once the link between the supporting elements has been established, the premises/details related to evidences are parsed and stored in an array. In a similar way, the requirements  $p$  and sub-requirements  $sp$ , modelled as practices, are extracted from requirements plugin, as shown in Algorithm 2.

**Algorithm 1** Extracting elements from process models

```

1: procedure EXTRACT PROCESS ELEMENTS( $Proc$ )
2:    $Q \leftarrow \emptyset$   $\triangleright Q$  is the queue of non-visited elements
3:    $V \leftarrow \emptyset$   $\triangleright V$  is the queue of visited elements
4:    $Q \leftarrow Q \cup \text{getTopLevelElements}(Proc)$ 
5:   for all  $e \in Q$  do
6:      $V \leftarrow V \cup \{e\}$ 
7:      $Q \leftarrow Q \setminus \{e\}$ 
8:      $E_{breakdownElement} \leftarrow \text{getLinkedElements}(e)$ 
9:     for all  $le \in E_{breakdownElement}$  do
10:      if ( $evidenceDeatail(le) \neq \emptyset$ ) then
11:        split the evidences and
12:        add in array  $evi$ 
13:      end if
14:    end for
15:  end for
16: end procedure

```

**Algorithm 2** Extracting requirements from requirements plugin

```

1: procedure EXTRACT REQUIREMENTS( $Reqs$ )
2:    $R \leftarrow \emptyset$   $\triangleright R$  is the queue of non-visited elements
3:    $T \leftarrow \emptyset$   $\triangleright T$  is the queue of visited elements
4:    $R \leftarrow R \cup \text{getRequirementPractices}(Reqs)$ 
5:   for all  $p \in R$  do
6:      $T \leftarrow T \cup \{p\}$ 
7:      $R \leftarrow R \setminus \{p\}$ 
8:      $P_{subRequirements} \leftarrow \text{getSubRequirements}(p)$ 
9:     for all  $sp \in P_{subRequirements}$  do
10:      if ( $practiceAdditionalInfo(sp) \neq \emptyset$ ) then
11:        split the requirements and
12:        add in array  $req$ 
13:      end if
14:    end for
15:  end for
16: end procedure

```

Algorithm 3 illustrates the detecting fallacies procedure. In this context, the requirements (briefDescription) related to a specific element (e.g., role) are matched with the provided information details (e.g., Skills or MainDescription). If the evidence details are omitted or the rationale is not provided, it means that process contains the omission of key evidence fallacies. The approach provides validation results including the list of



elements containing sufficient and insufficient information (i.e., detected fallacies). In addition, the appropriate recommendations to resolve the specific deviations are presented. These results can be printed on the console, or otherwise the validation reports are generated in the selected folder. Figure 82 shows the process model and the validation result, including the list of roles containing sufficient information (enclosed in green box), omitted details of evidences, and recommendations (enclosed in red box). The results have been printed on the console, or otherwise the fallacies reports would have been generated.

---

**Algorithm 3** Detecting omission of key evidence fallacies

---

```
1: procedure DETECTING FALLACIES(Reqs, Proc)
2:   getEvidenceDetails from process
3:   (call Algorithm 1)
4:   getAdditionalInfo from requirements
5:   (call Algorithm 2)
6:   for all le  $\leftarrow$  evi do
7:     for all sp  $\leftarrow$  req do
8:       match (sp.name, le.name)  $\wedge$  (req[j], evi[i])
9:       if (sp.name = le.name)  $\wedge$  (req[j] = evi[i])
10:        then returns true
11:      else
12:        returns false
13:        generate omitted information
14:        and provide recommendations
15:        regarding evidences
16:      end if
17:    end for
18:  end for
19: end procedure
```

---

Process engineers and/or safety engineers then modify the process by providing required evidences or rationale for omitted information, the revised version will be revalidated again. Therefore, it ensures that the final argumentation that will be generated from the process (which is specified in the next step) is valid. Once fallacies are eliminated, by modifying the process models, and rerunning the checking process yields no further flaws, the process-based arguments are generated from the modified models.



The screenshot displays the 'ECSS-E-ST-40 Software Design and Implementation Planning' tool. It features a table with columns: Presentation Name, Model Info, Team, Type, Planned, Multipl..., and Optional. The table lists various process elements, including 'ECSS-E-ST-40\_Planning\_Phase\_Software\_Design\_and\_Implementation', 'ECSS-E-40\_Planning\_Process', 'design of software items', 'coding and testing', 'Develop and Document Software Units', 'AOCS AIT Engineer', 'AOCS Engineer', 'AOCS SW Architect', 'AOCS SW V&V Manager', 'Development Team Leader', 'Test Software Units', and 'integration'. The 'Planned' column has checkboxes, and the 'Optional' column has checkboxes.

Below the table, there is a 'Validation Report [Eclipse Application] C:\Program Files\Java\jre1.8.0\_121\bin\javaw.exe (Jun 21, 2018, 3:59:25 PM)'. The report contains the following text:

```

Certifications against below mentioned roles are insufficient:
1. AOCS ENGINEER.
   DETECTED FALLACIES: Certifications against following competencies/requirements are omitted:
   - University degree in engineering
   - Working experience with Linux, Matlab and Satsim
   RECOMMENDATION:
   - Add skill certifications against above omitted evidences for the AOCS ENGINEER 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 experience with Matlab/Simulink
   - Knowledge of design analysis and design test methodologies
   - Good analytical and problem-solving skills
   RECOMMENDATION:
   - Add skill certifications against above omitted evidences for the DEVELOPMENT TEAM LEADER to achieve sufficiency or provide rationale for its omission.
Certifications against below mentioned roles are sufficient:
3. AOCS AIT ENGINEER
4. AOCS SW ARCHITECT
5. AOCS SW V&V MANAGER

```

Figure 82. Result after detecting omission of key evidence fallacies

#### 5.9.1.4 Mapping between process elements and argumentation elements (\*)














To perform the generation of process-based arguments, the process models are described according to the UMA metamodel in EPF-C, whereas the argumentation models should be compliant to the CACM metamodel in OpenCert and should be rendered via a concrete syntax, e.g. GSN. The main mapping between these metamodels is described in Table 10. This mapping is coherent with what was initially conceived in the context of SafeCer project and published in [29].

In particular, within AMASS, the mapping is as follows: the ProcessComponent that contains the information of the process is mapped into a *Case*, whereas the Planning Phases are mapped into the *Claims* stating that the planned process is in compliance with the required standard-level. Specifically, the mapping is focused on the Work Breakdown Structure of processes in EPF Composer. These *Claims* are decomposed into *Sub-claims* by showing that all the *Activities* have been planned, in turn, for each *Activity* all the *Tasks/TaskDescriptors* have been planned are mapped into *Claims* in CACM (represented by Goal in GSN). *ArgumentReasoning* elements (aka GSN strategies) are created in order to divide the Claim into Sub-claims. The crucial process elements associated to a *TaskDescriptor*, namely requirements related to *RoleDescriptors*, *WorkProductDescriptors*, *Guidelines*, *ToolMentors* and *Tools*, *Checklist*, and *Example* are mapped to the Sub-claims. In case *WorkProductDescriptor* has been planned as expected output of a task, it would be mapped into undeveloped Claim. The property of the claim *toBeSupported* = "true" means that it requires further development. Evidences associated to these elements are mapped into *InformationElementCitation* typed "solutions" showing that the particular goals have been achieved. The rationale related to element (e.g., tool qualification is not needed since source code is fully tested) is mapped into *InformationElementCitation* typed "justification". The "purpose" attribute of the process is used to show that the process is compliant with the standard, which is mapped to *InformationElementCitation* "context".



The relationship *AssertedInference* (SolvedBy in GSN) links the TaskDescriptor (source) to a target element, which is not a solution, whereas *AssertedEvidence* (SolvedBy in GSN) links the source to a target element (i.e., solution). Finally, a relationship *Asserted* (inContextOf in GSN) relates the sub-goal related to the task with a piece of contextual information related to the standard to be considered.

**Table 10.** Mapping between process model, argumentation model and GSN

UMA Metamodel	CACM (Argumentation Metamodel)	GSN Concept
ProcessComponent 	Case 	
Phase 	Claim	Goal 
Activity 	Claim	Goal 
TaskDescriptor 	Claim	Goal 
Process purpose (Standard)	InformationElementCitation Property type = "context"	Context 
A set of RoleDescriptors   WorkProductDescriptors  Guidelines  ToolMentors and Tools  Checklist  Example 	ArgumentReasoning	Strategy 
WorkProductDescriptor (expected output) has been planned	Claim Property toBeSupported= "true"	Undeveloped goal 
Guidelines should be followed 	Claim Property toBeSupported= "true"	Undeveloped goal 
Evidences associated to: WorkProductDescriptor  RoleDescriptor  Guideline  ToolMentor and Tool  Checklist  Example 	InformationElementCitation Property type = "solution"	Solution 
Rationale related to RoleDescriptor/Tool Qualification	InformationElementCitation Property type = "justification"	Justification 
Relationships (between a TaskDescriptor and a role/tool/...)	AssertedInference (target is not a solution)	SolvedBy 
Relationships (between a TaskDescriptor and WorkProductDescriptor)	AssertedEvidence (target is solution)	SolvedBy 
Relationships (for Context)	Asserted	InTheContextOf 

The mapping is implemented by using Epsilon Transformation Language (ETL). ETL is a hybrid, rule-based model-to-model transformation language and provides the enhanced flexibility to transform many input to many output models. A plugin has been implemented in the AMASS Prototype P1 [20], which automatically



transforms the process model into safety argument fragments (i.e., model and diagram) using ETL. The generated argumentation model and diagram are visualized via the assurance editor in OpenCert. In deliverable, the mapping is enhanced. Figure 83 shows the generated argumentation model and the corresponding diagram, compliant to the CACM metamodel that are visualized in assurance case editor in OpenCert. The generated process-based arguments are free from the fallacies and provide valid justification that the evidences are sufficient to meet the standard's requirements.

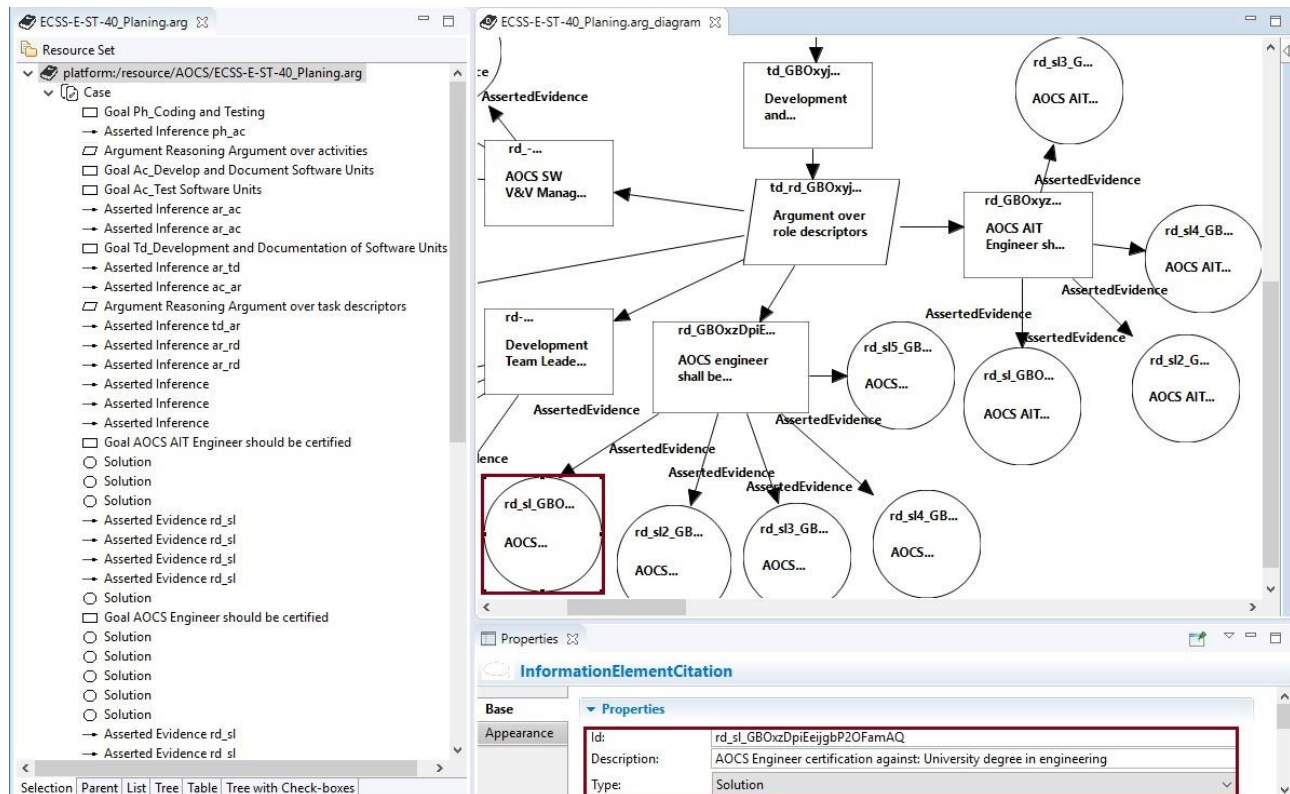
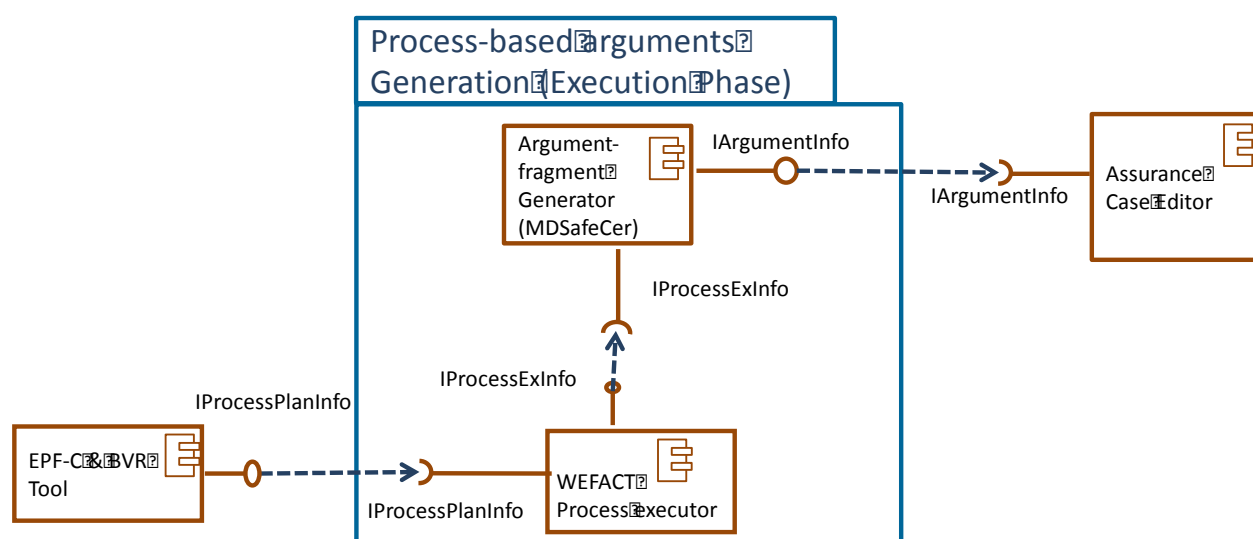


Figure 83. Generated argumentation model and diagram

## 5.9.2 Generating Process-based Argumentation Representing Executed Processes (\*)

During the execution phase, the design is slightly different. As depicted in Figure 84, based on a specific model regarding an executed process (managed by WEFACT), an argument is expected to be automatically generated by following MDSafeCer principles and visualized via the Assurance Case Editor. To perform such generation, a mapping similar to the one presented in Table 10 is proposed in what follows.



**Figure 84.** Process-based arguments generation (Execution Phase)

In the automatic generation of process-based arguments, arguing about compliance of executed processes, WEFACT will be used as a workflow tool for generating the evidences for the requirements from the applicable Security and Functional Safety Standards, and for delivering the respective input for the MDSafeCer Argument Fragment Generator.

Cross-domain re-use is possible between the argument fragments as well as the evidences in those cases where the similarity of the requirements from the standards allows their similar treatment in the argumentation.

The entire workflow of the Process-based argument generation runs as follows:

By means of the tool EPF-C, together with the BVR (Base Variability Resolution) tool, the process model for the applicable Security and Functional Safety standards of the domains under consideration is created and instantiated towards the company practices and the specific needs of the individual project. The resulting process model instantiation (IProcessPlanInfo) is stored in SPEM2.0 format and subsequently imported into WEFACT.

In WEFACT, the necessary steps, for delivering the evidences (also known as GSN solutions) for the process-based requirements, are created. This is done manually by assigning adequate processes for verifying the process requirements – with the option to automate this process as far as possible in a later project phase.

In addition, WEFACT creates the data for the argument generation (IProcessExInfo), which is linked to the aforementioned V-Plans and evidences. This data is the input for the Argument Fragment Generator (MDSafeCer), which creates the argument fragments corresponding to the process activities under consideration. The generation is performed via a model transformation that transforms the modelling elements in WEFACT into modelling elements of the argumentation.

The generation of arguments is done in the following 4-step workflow:

1. WEFACT reads the UMA file (possibly generated with EPF Composer) with the following assumptions:
  - The UMA file contains both (standards) requirements and processes for proving them.
  - The requirements-related text contains (before the first blank) the number of the respective clause in the standard.
2. The user defines the activities with the associated tools and the input/output directories in WEFACT. Note that this information is not directly represented in the AMASS metamodel but is later used by WEFACT for
  - executing the processes in order to generate the evidences (step 3), and
  - generating the argument fragments by composing the GSN argumentation elements (step 4).

3. Then, the processes are executed within WEFACT and evidences are generated.
4. Finally, WEFACT creates the arguments and the evidences by iterating all processes.
  - Arguments are written into an xml file in CACM-conformant format using data elements from WEFACT plus additional text. Due to the assumed 1:1 relation between them, both requirements and processes become GOALS. The STRATEGY is derived from the assigned tool, the JUSTIFICATION is composed of Users, Roles, the Tool Description and the Rationale of the Requirement. Finally, the SOLUTION is derived from the input/output Work products.
  - As foreseen by the CACM metamodel, the evidences are not assigned to individual solutions but all evidences are assigned to the Assurance Cases as a whole.

The resulting xml file is imported in OpenCert to become part of the AMASS metamodel instances.

The following table gives an overview on the mapping between WEFACT and OpenCert model elements.

**Table 11.** Mapping between WEFACT and OpenCert model elements

Wefact Model	CACM (Argumentation Metamodel)	GSN Concept
Project	Case	
Process	Claim	Goal
Requirement	Claim	Goal
Tools	ArgumentReasoning	Strategy
Link between Requirement and Process	AssertedEvidence	SolvedBy
Users / Roles	InformationElementCitation Property type= "justification"	Justification
Description of Tool / Rationale of Requirement	InformationElementCitation Property type= "justification"	Justification
WorkflowStatus	InformationElementCitation Property type="solution"	Solution
Artefacts (Input/Output files) → Work product	InformationElementCitation Property type="solution"	Solution

In WEFACT, the project itself represents what corresponds to the "Case" in CACM.

A clause number is specified at the beginning of the description of a requirement to identify its reference in a specific standard. The new function parses the text and identifies this clause number. The outputs of processes linked to a requirement represent a work product, so that several outputs can be combined to form one single work product. This is represented as "InformationElementCitation solution" in the table. Process and requirement are treated as "Claim". Consequently, the Link between Requirement and Process represents the "AssertedEvidence" mentioned above, which serves as a reference to the corresponding work product. As the WEFACT workflow supports a many-to-many relation between requirements and processes, this can become complex for creating an argument fragment. To stay conformant to the OpenCert platform, one process per requirement is used for the proof of concept. Every argument fragment corresponds to exactly one process.

Additionally, for each process, a data record identifying how the fulfillment has been proven is given (Tools). This corresponds to "ArgumentReasoning" in the CACM Model.

To justify any actions, the tool description and the rationale of a requirement execution are mapped to “InformationElementCitation justification”. In the current concept for implementing the approach, it is assumed that the status is set by a responsible human user.

The WorkflowStatus is transformed into Evidence model element instances. As output, WEFACT creates a CACM-conformant xml file. This file is imported by an existing standard importer, which can be started by clicking a button in the OpenCert editor.

## 5.10 Automatic generation of product-based arguments

This section provides the design solution for the automatic generation of product-based arguments. This solution represents a further development of what was presented in [51]. To perform the generation of argument fragments from component contracts, first the contracts (weak and strong) have been enriched with argumentation information, namely: context statements, claims and evidence artefacts. This information is provided by the contract and traceability editor. As documented in D3.2 [9]/D3.3 [10], and D3.5 [11], the status of the weak contracts is validated and used as the input to the argument generation. The Argument-fragment generator creates the argument-fragments in the corresponding assurance case project where they can be viewed in the assurance case editor. The generator uses a pre-existing argument pattern for the generation. The generated argument-fragments include only those contracts whose assumptions are validated, hence only those artefacts related to the validated contracts.

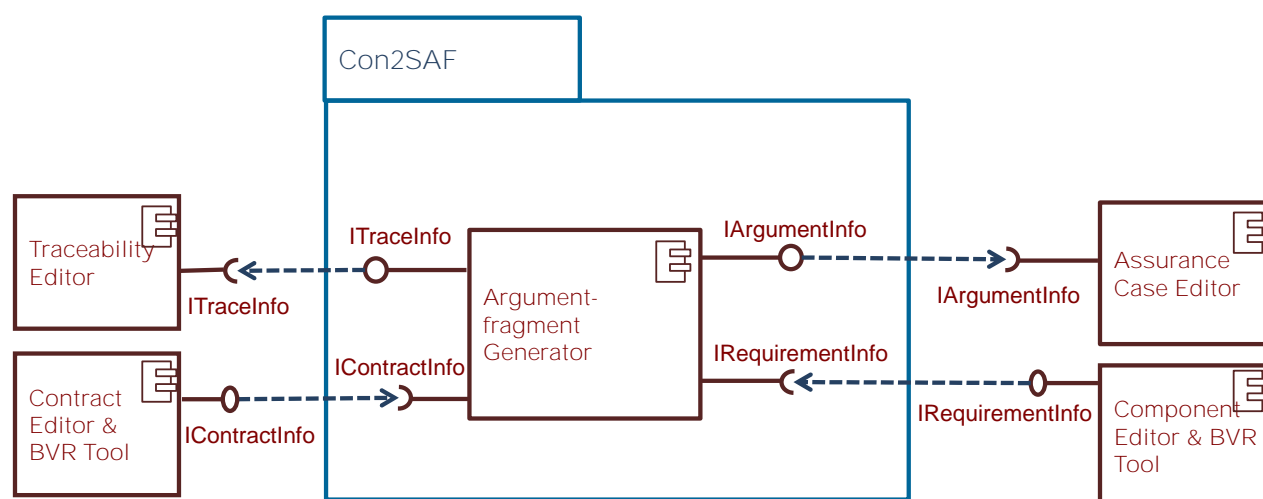


Figure 85. Product-based argument generation

This design solution has been further developed and presented at AdaEurope-2018 [156].

### 5.10.1 Argument-fragment generation at the architectural pattern level (\*)

Architectural patterns are used to capture the reusable reasoning regarding a design solution to a particular problem. The design solution exhibits certain properties that guarantee to address the targeted problem. Capturing the information about architectural patterns in assumption-guarantee contracts, by using the design pattern template detailed in D3.3 [10], can serve as the basis for assuring that the application of the architectural pattern adequately addresses the problem that it is trying to solve. The architectural patterns can be modelled in CHES using the component type element, and the design pattern template information can be captured using the component type assumption-guarantee contracts and the related assurance information elements.

To automate the argument-fragment generation, we developed the assurance pattern for assurance of architectural pattern application, which utilises the design pattern template information that can be captured in the CHES model. The assurance pattern is detailed in D3.3 [10].



The generation is done in the similar way as for the OCRA contracts using the Con2SAF component. The Argument-fragment Generator is extended to use the assurance pattern specific to architectural pattern application when the type of contracts specified in the CHES model corresponds to design pattern contracts. With this extension, the Argument-fragment Generator supports argument-fragment generation for both OCRA and design pattern contracts, where the first are formal, and the latter are informal contracts.



## 6. Implementation Solution for Cross/intra domain reuse: a way forward

In this chapter, a way forward concerning the implementation is proposed. The real implementation is expected to be given in D6.6 [21], output of Task 6.3. Table 12 and Table 13 provide details concerning the three main functionalities for the AMASS Cross/Intra Domain Vision.

**Table 12.** STO4-Reuse Assistant as well as Product/Process/Assurance Case Line Specification + semi-automatic generation of arguments

ID	Short Description	Description	Prototype N°	Priority	Elaborated in section
WP6_RA_001	Intra-Domain, Intra standard, Reuse Assistance	The AMASS tools shall enable partial reuse of compliance artefacts when transiting from one project to another (different criticality level, etc.). The commonality that characterizes the different projects should be recognized and proposed as reusable process structure.	P1	Shall	5.2 as well as 5.3.2
WP6_RA_002	Intra-Domain, Cross standards, Reuse Assistance	The AMASS tools shall enable partial reuse of compliance artefacts when transiting from one project to another (different/same criticality level, if applicable, but different standards (e.g., AutomotiveSPICE, ISO 26262).) The commonality that characterizes the different projects should be recognized and proposed as a reusable process structure.	P1	Shall	5.2 as well as 5.3.2
WP6_RA_003	Intra-Domain, Cross versions, Reuse Assistance	The AMASS tools shall enable partial reuse of compliance artefacts when transiting from one project to another (different/same criticality level, if applicable, but different standards (e.g., ISO 26262-2011, ISO 26262-2018).) The commonality that characterizes the different projects should be recognized and proposed as reusable process structure.	P1	Shall	5.2 as well as 5.3.2
WP6_RA_004	Cross-Domain Reuse Assistance	The AMASS tools shall enable partial reuse of compliance artefacts when transiting from one project to another belonging to different domains (e.g., from automotive to avionics). The commonality that characterizes the different projects should be recognized and proposed as reusable process structure.	P1	Shall	5.2 as well as 5.3.2
WP6_RA_005	Intra-Domain, Intra standard, Different Stakeholders, Reuse/Integration Assistance	The AMASS tools shall enable partial reuse of compliance artefacts during the integration (manufacturer/supplier). Assumed process requirements vs. actual process requirements.	P1	Shall	5.2 as well as 5.3.2



**Table 13.** STO4-Product/Process/Assurance Case Line Specification + semi-automatic generation of arguments

ID	Short Description	Description	Prototype Nº	Priority	Elaborated in section
WP6_PPA_001	The AMASS tools must support variability management at process level	The system shall enable users to specify what varies (and what remains unchanged) from one process and its family members.	P1	Shall	5.3.2
WP6_PPA_004	The AMASS tools must support specification of variability at the component level	The system shall enable users to specify what varies (and what remains unchanged) from one component and its family members (e.g., its evolved versions at component level).	P1	Shall	5.3.3
WP6_PPA_005	The AMASS tools must support variability management at the assurance case level	The system shall enable users to specify what varies (and what remains unchanged) from one assurance case and its family members.	P2	Shall	5.3.4
WP6_PPA_002	Semi-automatic generation of product arguments	The system should reduce efforts of manual creation of product-based assurance case arguments. This could be done by enabling semi-automatic generation of product-based arguments-fragments.	P1	Shall	5.9
WP6_PPA_003	Semi-automatic generation of process arguments	The system shall reduce efforts of manual creation of process-based assurance case arguments. This could be done by enabling semi-automatic generation of process-based arguments-fragments.	P1	Shall	5.10
WP6_RA_003	Reusable off the shelf components	The system shall provide the capability for reuse of pre-developed components and their accompanying artefacts.	P1	Shall	Included in D3.2 [9] and D3.3 [10]

The requirements regarding metrics (WP6\_RA\_007, WP6\_RA\_008 and WP6\_RA\_009) do not require an implementation and are addressed in Chapter 14.

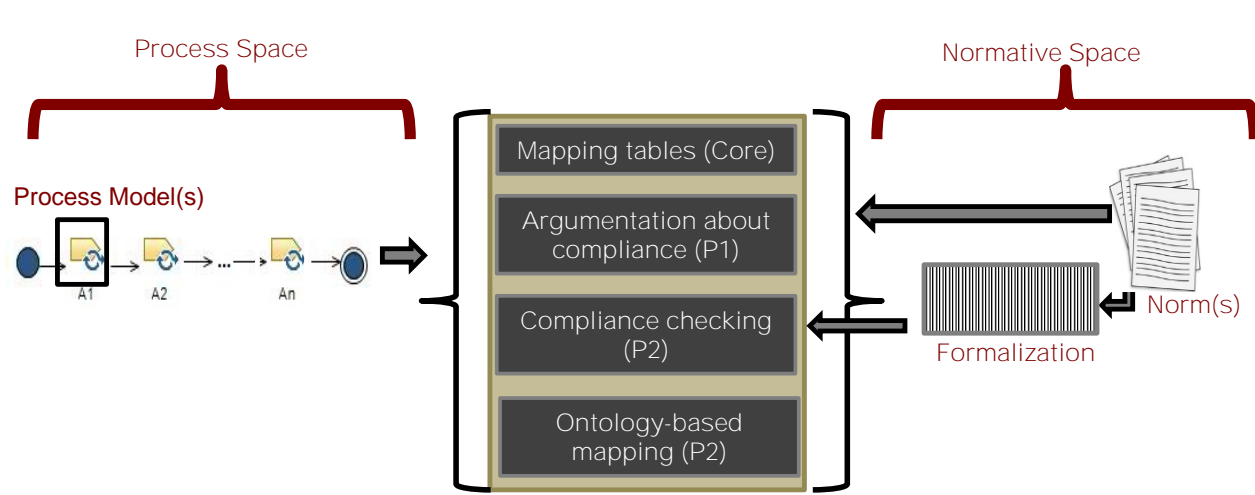
## 7. AMASS vision for compliance management (\*)

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), or 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: 1) properly interpret and model the requirements coming from the standards, 2) provide the evidence expected to substantiate such requirements, and 3) provide the relation between requirements and evidence. The vision of AMASS for compliance management is exemplified in Figure 86. As this figure depicts, compliance management is meant as management of 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 for 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.

Thus, four different or complementary methods are developed/enhanced and integrated. These four methods are all expected to compare the normative space with the process space:

- Compliance table generation, as it was presented in D6.1 [17], constitutes an OPENCOS result that has been integrated in the first release of the AMASS platform (see D2.2 [4] for architectural details). Compliance table generation consists of the generation of a table representing a checklist. Compliance tables, however, as explained in [155], can be modelled also in EPF Composer.
- (Generation of) argumentation about compliance, as it was presented in D6.1, constitutes a SafeCer conceptual result, which was initially implemented as a proof of concepts within the WEFACT tool (see [29]). Within AMASS, the generation of arguments targeting compliance has been implemented and integrated within P1 (the second iteration of the AMASS prototype), based on the designed solution provided in Section 5.4 of this document. As explained in Section 5.9, such functionality has been empowered by designing the fallacy-free generation of process-based argumentation to be implemented for P2. (Generation of) argumentation about compliance consists of the generation of (counter) arguments explaining why compliance is or is not achieved.
- (Automatic) compliance checking represents a novelty. None of the previous projects have considered this opportunity. (Automatic) compliance checking is aimed at offering increased efficiency and confidence via the adoption of logic-based reasoning for checking compliance.
- Ontology-based mapping, as it was presented in D6.1 [17], constitutes a SafeCer conceptual result, which was initially implemented as a proof of concepts. Within AMASS, the principles of this method have been incorporated, but its design has been slightly changed since a different language for representing the ontology has been selected. The re-designed method is expected to be implemented and integrated within P2 (the third iteration of the AMASS prototype), based on the designed solution provided in Chapter 12.



**Figure 86.** Compliance Management Vision, adapted from [155].

Chapters 8÷11 presents the method regarding compliance checking. First, the conceptual solution is provided (Chapter 8), then, possible design solutions are discussed and compared (Chapter 9); then the AMASS vision for compliance checking in AMASS is designed (Chapter 10). Finally, the way forward for the implementation is sketched (Chapter 11).

Chapter 12 develops the method regarding ontology-based mapping and Chapter 13 explains the corresponding way forward for its implementation.

## 8. Conceptual solution for compliance checking

In this chapter, the conceptual solution for the compliance checking vision is elaborated. This conceptual solution is built on top of the state-of-the-art theoretical foundation. Thus, in Section 8.1, some essential background information is provided. Then, in Section 8.2, initial manual explorations of its usefulness in the AMASS context are documented. Finally, in Section 8.3, the needed modelling and analysis capabilities for enabling compliance checking are identified.

### 8.1 Essential Background information

This section is structured as follows. In Section 8.1.1, Defeasible logic is presented. In Section 8.1.2, Formal Contract Logic is presented. Finally, in Section 8.1.3, Regorous Process Designer is presented.

#### 8.1.1 Defeasible Logic

This subsection is structured as follows. In Section 8.1.1.1 the basis of defeasible logic is given. In Section 8.1.1.2., a defeasible logic derivation called defeasible deontic logic is presented.

##### 8.1.1.1 Defeasible logic: the basis

Defeasible Logic is a non-monotonic formalism proposed by Nute [54]. Non-monotonic reasoning [55] is an approach able to capture different and sometimes incompatible facets of reasoning, where a larger set of initial “assumptions” does not necessarily imply a larger set of “consequences”. The understood notion is that the derived conclusions are tentative and can be retracted once new information is available. However, logics created for non-monotonic reasoning have high computational complexity, i.e., default logic [56], which proposes a compact representation of information, in which default prevails where more information is required for decision-making. Some derivations of default logic are justified: default logic [57], disjunctive default logic [58] [59], contrived default logic [60], and rational default logic [61]. Other non-monotonic logics are auto-epistemic logic [62], which permits the representation of agent’s beliefs allowing the drawing of some conclusions based on these beliefs, and circumscription logic [63], which represents common sense reasoning to draw conclusions of a theory by minimizing its models to those that have minimal extensions in the universe of models. Defeasible Logic is an alternative for non-monotonic reasoning and it is considered a logic that is “tunable” to the environment. Defeasible logic also preserves low computational complexity [64]. The underlying notational convention for defeasible logic is as follows:

- There is a set of proposition literals.
- If  $q$  is a literal,  $\neg q$  is its complement.
  - If  $q$  is a positive literal  $p$ , then  $\neg q$  is  $\neg p$ .
  - If  $q$  is a negative literal  $\neg p$ , then  $\neg q$  is  $p$ .
- There is a set of modal operators.
- Rules are defined over a language (or signature)  $\Sigma$ .
- There is a set of propositions (atoms).
- There are labels that may be used in the rule.

The base knowledge in defeasible logic is called defeasible theory. It is represented by the triple  $(F, R, >)$ , where [65]:

- $F$  and  $R$  are finite sets of facts and rules.
- $>$  is a superiority relation of  $R$ .

Facts are logical statements describing indisputable statements representing the state of affairs or actions that have to be performed. A fact is considered always true. Rules are binary relations between a set of literals (the antecedent  $A(r)$ , which can be empty) and a literal (the consequent  $C(r)$ ). Formally,

$$r: A(r) \hookrightarrow C(r)$$

where

- $r$ : is a unique identifier of the rule.
- $A(r) = a_1, \dots, a_n$  is the antecedent of the rule.
- $\hookrightarrow: \{\rightarrow, \Rightarrow, \text{and } \rightsquigarrow\}$ , representing strict, defeasible and defeater rules.
- $C(r)$ : the consequent (or conclusion) of the rule.

Defeasible logic supports three kinds of rules:

- a) **Strict rules**: rules in the classical sense, whenever the premises are indisputable, so is the conclusion.
- b) **Defeasible rules**: rules that can be defeated by contrary evidence; defeasible rules with empty antecedent can be considered as a presumption.
- c) **Defeaters**: are special kind of rules. They are used to prevent conclusions, not to support them.

Strict rules with empty antecedent are a way to model facts. Facts are more likely to be used to describe contextual information while strict rules are more likely to be used to model the reasoning underlying the context. Defeasible logic is a skeptical logic, i.e., it does not support contradictory conclusions. Additionally, defeasible logic seeks to resolve conflicts. The complete proof theory for DL is sketched in [65].

### 8.1.1.2 Defeasible Deontic Logic

Defeasible deontic logic is a derivation of defeasible logic that addresses deontic modes [66] by extending deontic logic. Deontic modes or modes of obligations [67] are concepts such as the obligatory (that we ought to do), the permitted (that we are allowed to do), and the forbidden (that we must not do). Deontic Logic extends first-order logic with the deontic operators (O for obligations, P for permissions, and F for prohibitions). Defeasible deontic logic [68] supports deontic operators in the modelling of norms by satisfying the following equivalence relations:

$$OA \equiv \neg P \neg A \quad \neg O \neg A \equiv PA \quad O \neg P \equiv FA \quad \neg PA \equiv FA$$

The following relation  $OA \rightarrow PA$  is also supported, and it means that if A is obligatory, then A is permitted. In addition, there are cases in which breaches of norms can be supported by the normative system with the use of special secondary norms. These special norms are policies that are included to express the respective obligations for the actors to compensate the mentioned breaches. In deontic logic, this type of expression, namely, the activation of certain obligations in case of other obligations have been violated is referred to as contrary-to-duty obligations (CDT) [69]. A CDT is not a usual conflicting obligation, which overrides a primary obligation. CDT norms can be logically thought of as a special exception of primary obligations. For modelling a CDT rule, the symbol  $\otimes$  (which is called the reparational connector) is used. In this sense, a norm with an exception can be represented as:

$$OA \Rightarrow OB \otimes OC$$

The previous expression is called “*reparation chain*”. The meaning of a reparation chain is that B is obligatory, given the obligation of A, but if the obligation of B is violated, i.e. we have not B, then the violation is compensated by C (which is then obligatory).

## 8.1.2 Formal Contract Logic

In this section, Formal Contract Logic (FCL) is presented. In Section 8.1.2.1, the basis of FLC is given. In Section 8.1.2.2 the types of rules used in FCL are presented.

### 8.1.2.1 FCL: the basis

FCL [68] is a formal representation language designed in the context of business management to formalize the information that is present in contracts. FCL has also been used for analysing the compliance of business processes against regulatory requirements. FCL underlying logic is constituted by the combination of

defeasible logic (introduced in Section 8.1.1.1) and defeasible deontic logic (Introduced in 8.1.1.2). In FCL a norm is represented by a rule:

$$a_1, \dots, a_n \Rightarrow c$$

Where:

$a_1, \dots, a_n$ : Conditions of applicability of the norm/rule.  
 $c$ : Normative effect of the norm/rule.

FCL differentiates two normative effects i.e. the definition of a new term and the triggering of deontic notions. Deontic notions, as seen in Section 8.1.1.2, are the obligations, permissions and prohibitions. Obligations and prohibitions are constraints that limit the space of action of a process. The difference from other types of constraints is that they can be violated, but a violation does not imply an inconsistency if the violation can be compensated. Process with compensated violations is still compliant [71]. Permissions cannot be violated. Thus, permissions do not play a direct role in compliance. Permissions can be used to determine that there are neither obligations nor prohibitions to the contrary or to derive other deontic effects. Legal reasoning and legal theory typically assume a strong relationship between obligations and prohibitions [72]: the prohibition of  $A$  is the obligation of  $\neg A$  (the opposite of  $A$ ), and then if  $A$  is obligatory, then  $\neg A$  is prohibited. For this reason, both terms, obligations and prohibitions, are subsumed under the term obligations. Compensations of violations are implemented based on the notion of *reparation chain* [69], which has an expression of the form:

$$O_1 c_1 \otimes O_2 c_2 \otimes \dots \otimes O_n c_n$$

Where:

$O_i$ : is an obligation  
 $c_i$ : is the content of the obligation

FCL is equipped with a binary relation over rules that allow handling rules with conflicting conclusions. A conflicting conclusion appears when there is a rule  $r$  that sets a general prohibition and a second rule  $r'$  that derogates the prohibition, permitting the conclusion. This type of situations is common in legal reasoning and can be modelled by saying that  $r'$  is “stronger” than  $r$ :

$$r' > r$$

If both rules apply, the superiority relation  $>$  defines that  $r'$  defeats  $r$ , and there is not conflict any more. An obligation  $[O]p$  is derivable if:

- $[O]p$  is given as one of the facts
- There is a rule:

$$r: a_1, \dots, a_n \Rightarrow [O_1]p_1 \otimes \dots \otimes [O_m]p_m$$

Such that:

- a) For all  $1 \leq i \leq n$ ,  $a_i$  is provable, and
- b) For all  $1 \leq j \leq m$ ,  $[O_j]p_j$  and  $\neg p_j$  are provable, and
- c) For all rules

$$s: b_1, \dots, b_k \Rightarrow [D_1]q_1 \otimes \dots \otimes [D_l]q_l \otimes [D]p'$$

Such that  $p'$  is the negation of  $p$ , either:

- i. Exist  $1 \leq i \leq k$  such that  $b_i$  is not provable, or
- ii. Exist  $1 \leq j \leq l$  such that either  $[D_1]q_1$  or  $\neg q_j$  is not provable, or
- iii.  $r$  defeats  $s$

The idea is that there must be a rule that fires (firing rules are rules that provides the conditions for triggering other rules [161]), so all the elements in the antecedent are provable (a), and in case the conclusion is an obligation for a reparation, all the obligations before have to be violated. Thus, the violated obligation was in force (thus the obligations were provable) and there is evidence that it was violated (thus, the negation of the content of each violated obligation is provable) (b). In addition, it is necessary to ensure that there are not rules for the opposite that fire (c), and if they do, these rules are weaker than the rule of the obligation to



conclude. For permission, there are the same conditions. In addition, if there is [O] $p$ , it is possible to conclude [P] $p$ . A complete representation of the logic is presented in [70] and [73]. FCL is agnostic about the nature of the literals it uses. They can represent tasks or other kind of process elements.

### 8.1.2.2 Types of rules in FCL

An important aspect of the study of obligations is to determine when they are in force (the lifespan of an obligation) and their implications on the activities carried out in a process [72]. An obligation can be *punctual* when it is in force for a particular time point; otherwise, an obligation is *persistent*. A persistent obligation remains in force until it is terminated or removed. For persistent obligations, it is possible to ask if they have to be obeyed for all instants in the interval in which they are in force, *maintenance obligations*, or whether doing or achieving the content of the obligation at least achieved once is enough to fulfil it, *achievement obligations*. Achievement obligations have another aspect: they can be *preemptive* if the obligation could be fulfilled even before the obligation is actually in force. Otherwise they are *non-preemptive*. Since obligations can be violated, the effects of the violations are also required. If the obligation persists after being violated, it is a *perdurant obligation*, if it does not, it is a *non-perdurant obligation*. The notation for obligations and permissions in FCL is the following:

- [P] $p$ :  $p$  is permitted
- [OM] $p$ : there is a maintenance obligation for  $p$
- [OAPP] $p$ : there is an achievement preemptive and perdurant obligations for  $p$
- [OAPNP] $p$ : there is an achieve preemptive, and non-perdurant obligation for  $p$
- [OANPP] $p$ : there is an achievement, non-preemptive and perdurant obligation for  $p$
- [OANPNP] $p$ : there is an achievement, non-preemptive, non perdurant obligation for  $p$

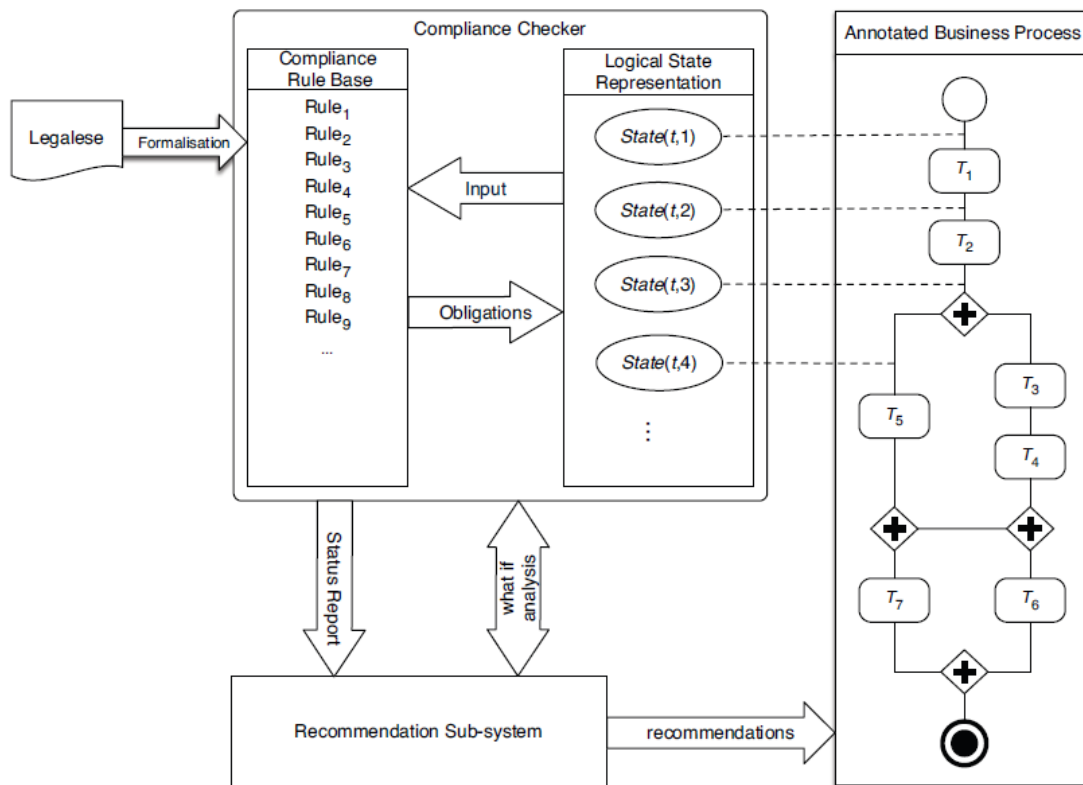
## 8.1.3 Regorous Process Designer

In this section, Regorous Process Designer (alias Regorous), a compliance checker created for business process compliance, is introduced. In Section 8.1.3.1, the methodology underlying Regorous is given. In Section 8.1.3.2, the architecture of Regorous is presented. In Section 8.1.3.3, SPINdle, the defeasible reasoner used by Regorous is presented.

### 8.1.3.1 Regorous Process Designer: the basis

Regorous Process Designer (for simplicity called only Regorous) [74] is a business process compliance checker, which is part of the of the NICTA's Regorous Tool suite<sup>16</sup>. It assists business analysts during the "*Process design phase of the business 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 [75] and [76]. 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 [72] requires an annotated process model and the formal representation of the regulation (See Figure 87). 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. For the representation of the process models, Regorous uses Business Process Management Notation (BPMN) [77], and the rules are modelled using FCL (see Section 8.1.2) to conform the compliance rule base.

<sup>16</sup> Regorous Process Designer is available under an evaluation license <http://www.regorous.com>



**Figure 87.** Abstract framework of Regorous, taken from [72]

To annotate the process model, semantic annotations (the process of attaching additional information to various concepts) are used. Semantic annotations, unlike classic text annotation (used by humans to read associated information), are used by machines to refer and compute information [78]. Regorous uses the semantic annotations to perform the analysis of compliance [79]. In FCL, the semantic annotations are literals, representing the effects of the tasks. Thus, for the  $n$ -th element in a trace (sequence of tasks, in which a process can be executed) it is used  $State(t,n)$  to semantically annotate the set of facts in the computation to determine which rules fire, and consequently which obligations are in force in  $Force(t,n+1)$ . In addition, the semantic annotation  $Force(t,n)$  contains the obligations that are in force but are not terminated in  $n$ . An obligation can be terminated if the deadline is reached, the obligation has been fulfilled, or if the obligation has been violated and it is not perdurant (as explained in Section 8.1.2.2, a perdurant obligation is an obligation that persist after being violated). A process is fully compliant if all its traces are compliant (all obligations have been fulfilled, or if violated, they have been compensated). A process is partially compliant if there is at least one trace that is compliant. To check compliance of an annotated process model against a relevant normative system, the procedure executed is the following [72]:

1. Generate an execution trace of the process.
2. Traverse the trace:
  - For each task in the trace, cumulate the effects of the task. Remark: if an effect in the current task conflicts with a previous annotation, update using the effects of the current task.
  - Use the set of cumulated effects to determine which obligations enter into force at the current task. This is done by a call to of FCL reasoner.
  - Add the obligations obtained from the previous step to the set of obligations carried over the previous task.
  - Determine which obligations have been fulfilled, violated or a pending, and if there are violated obligations, check whether they have been compensated.
3. Repeat for all traces.

### 8.1.3.2 Regorous Architecture

Regorous architecture is depicted in Figure 88. For the representation of the process models, it uses the *Eclipse Activiti BPMN 2.0* extended with features to allow users to add semantic annotations to the tasks in the process model. BPMN notation was selected because it has a graphical interface that is widely accepted in industry, and BPMN models can be translated to executable process models [75].

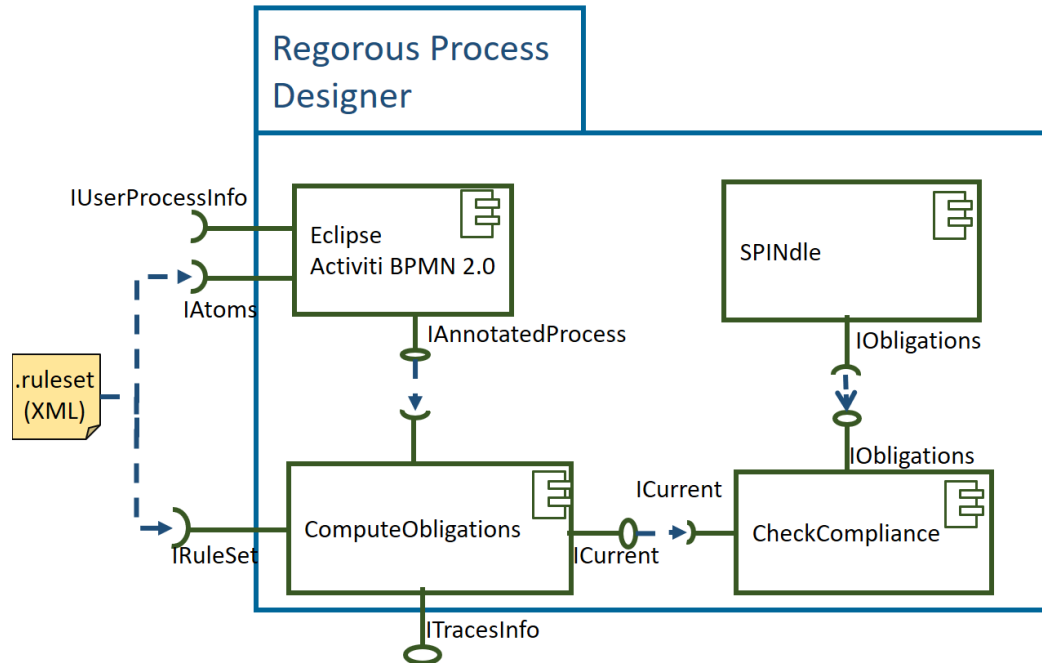


Figure 88. Regorous architecture

The procedure for compliance checking is based on two algorithms (See Figure 88), *ComputeObligations* and *CheckCompliance* [75]. *ComputeObligations* is the algorithm to determine the active chains (*current*). Given a set of literals  $S$  corresponding to effects of a task  $T$  in a process model, the algorithm *ComputeObligations* determines the current set of obligations for the process *Current*. The set of current obligations includes the new obligations triggered by the task as well as the obligations carried out from previous tasks. The algorithm *CheckCompliance* scans all elements of *Current* against the set of literals  $S$ , and determines the state of each reparation chain ( $C = A_1 \otimes A_2$ ) in *Current*. More specifically, the algorithm *CheckCompliance* scans all active chains one by one. Then for each of them it reports the status. For each chain in *current*, it looks for the first element of the chain and it determines the content of the obligation (so if the first element is OB, the content of the obligation is B). Then it checks whether the obligation has been fulfilled (B is in the set of effects), or violated ( $\neg B$  is in the set of effects) or simply we cannot say anything about it (none of B and  $\neg B$  is in the set of effects). In the first case the obligation is discarded and the chain removed from the set of active chains (similarly if the obligation was carried over from a previous task, i.e., it was in the set *unfulfilled*). In case of a violation, the information about this violation is added to the system. This is done by inserting a tuple with the identifier of the chain in the set *Violated*. Additionally, we know that violations can be compensated; thus, if the chain has a second element, the violated element is removed from the chain and the rest of the chain is put into the set of active chains. Finally, when the set of effects does not tell if the obligation has been fulfilled or violated, the obligation is propagated to the successive tasks by putting the chain into the set *Unfulfilled*. The algorithm also checks whether a chain/obligation was previously violated but then *Compensated*. *CheckCompliance* uses the SPINdle rule engine [80] for the evaluation of the FCL rules. Regorous has been implemented on top of Eclipse, specifically the version 3.7.2 which is called Eclipse indigo, which was available in June 2011.

### 8.1.3.3 SPINdle

SPINdle<sup>17</sup> [80] is an open source Java-based defeasible logic reasoner capable to perform efficient and scalable reasoning on defeasible logic theories (including theories with over one million rules). SPINdle consists of five components (depicted in Figure 89): the rule parser, the rule loader, the theory normalizer, the inference engine, and the I/O interface.

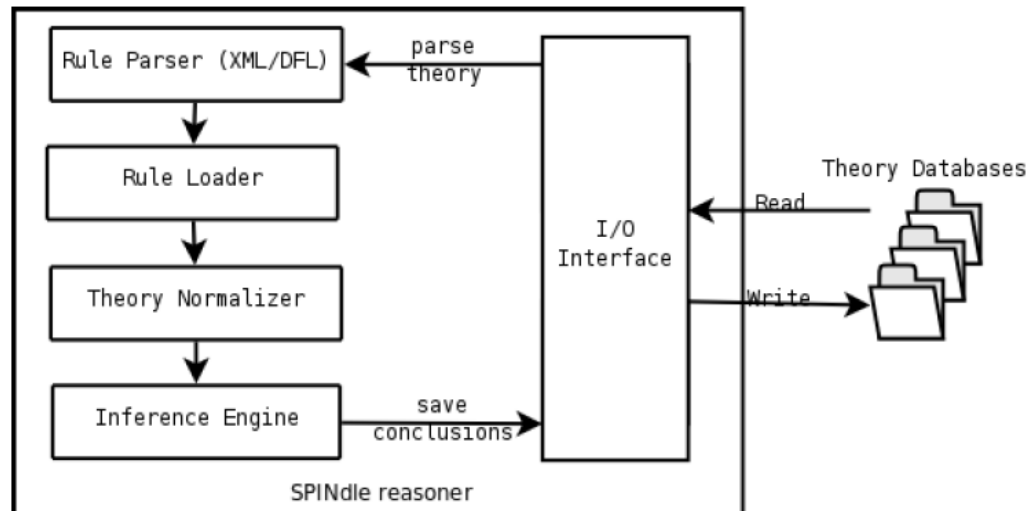


Figure 89. Main components of SPINdle reasoner. Taken from [80]

SPINdle accepts defeasible logic theories given in XML or plain text (with pre-defined syntax). The *rule parser* is used to transform the theory from a saved theory document into a data structure that can be loaded with the *rule loader*. The *theory normalizer* transforms the loaded theory into an equivalent theorem without superiority relations and defeaters, which helps to simplify the reasoning process in the *inference engine*. The *inference engine* generates the conclusions based on a series of theorem transformations in which it is asserted whether a literal is provable or not (and the strength of the derivation). The *I/O interface* module provides methods for helping users to load and save (modified) theory (also the derived conclusion) to and from the database. Theories can also be exported using XML. SPINdle was tested for correctness, scalability and performance using different forms of theorems<sup>18</sup>.

### 8.1.4 Property Specification Patterns for Finite-State Verification (\*)

A property specification pattern [95] is a generalized description of a commonly occurring requirement on the permissible state/event sequences of a finite model of a system. A pattern comprises a name, a precise statement of the structure of the behaviour described (pattern's intent), mapping into common specification formalisms, examples of known uses and relationships to other patterns. In property specification patterns, capital letters (e.g., P, Q, R, S) stand for stating formulas in state-based formalisms, such as derivatives of temporal logics. Each pattern has a scope (see Table 14), which is the extent of the program execution over which the pattern must hold.

Table 14. Property Specification Patterns Scope

Scope	Description
Global	The entire program execution
Before	The execution up to a given state

<sup>17</sup> SPINdle can be download freely from <http://spindle.data61.csiro.au/spindle/download.html>. Examples of the use of SPINdle can be seen in <http://spindle.data61.csiro.au/spindle/demo.html#clearReset>.

<sup>18</sup> More information about SPINdle can be found in <http://spindle.data61.csiro.au/spindle/documentation.html>

After	The execution after a given state
Between	Any part of the execution from one given state to another given state
After-until	Like between but the designated part of the execution continues even if the second state does not occur.

The patterns are created in a system organized into one or more categories, as depicted in Figure 90.

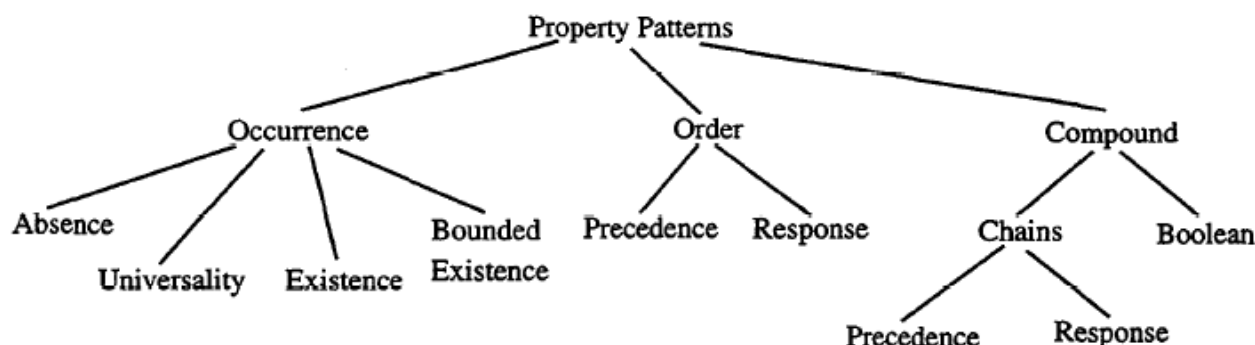


Figure 90. Property Specification Patterns Hierarchy

**Occurrence patterns** include:

- Absence: A given state does not occur within a scope.
- Existence: A given state must occur within a scope.
- Bounded existence: A given state must occur k times within a scope.
- Universality: A given state occurs throughout a scope

**Ordering patterns** include:

- Precedence: A state P must be always be preceded by a state Q within a scope.
- Response: A state P must be always be followed by a state Q within a scope.

**Compound patterns** include:

- Chain precedence: sequence of states, P1, ... , Pn must always be preceded by a sequence of states Q1,..., Qm.
- Chain response: sequence of states, P1, ... , Pn must always be followed by a sequence of states Q1,..., Qm.

### 8.1.5 EPF Composer metamodels (\*)

EPF Composer [44] is an open-source tool aiming at supporting the modeling of customizable software processes. We recall two open source standards used by EPF Composer and also required in this paper:

- UML 2.0 Diagram Interchange Specification is a standard that supports diagram interchange among modeling tools. Elements of interest are: *Activity*, which represents the process, *Node*, which represents a point in the process, and *Edge*, which connects points. Nodes can be of different types. An *Activity Parameter Node* represents a task. *Initial* and *Final Nodes* represent the start and the end of the process. *Fork* and *Join Nodes* represent the parallel flows and *Decision* and *Merge Nodes* represent conditional behavior.
- UMA (Unified Method Architecture) Metamodel [98], a subset of SPEM 2.0, is used to model and manage reusable method content and processes. We recall some required elements. *Method Content* defines the core elements, i.e., *tasks*, *work products* and *roles*. *Managed Content* defines textual descriptions, such as *Concept* and *Reusable Asset*. *Custom Category* defines a hierarchical indexing to manage method content. A *delivery process* describes a complete and integrated approach for performing a specific project and it contains a *Breakdown Structure*, which allows nesting of tasks.

### 8.1.6 Regorous metamodels (\*)

An FCL rule set is represented in the schema called *Combined Rule Set* from which we recall some elements. *Vocabulary* contains an element called *term*, which attribute *atom* is used to describe rule statements. The second element, called *Rule*, is used to define every rule of the logic. A rule is specified with the unique identifier called *label*, the description of the rule called *control objective*, and the actual rule called *formal representation*.

Regorous current implemented tool uses an open source engine which is based on BPMN 2.0 (Business Process Model and Notation). However, for checking compliance, Regorous translates the BPMN 2.0 description into the *Canonical Process Format (CPF)* [99], a modeling language agnostic representation that only describes the structural characteristics of the process. We recall the required CPF elements. A *Canonical Process* is the container of a set of *Nets* which represent graphs made up of *Nodes* and *Edges*. Nodes types can be (*OR*, *XOR*, *AND*) *Splits/Joint*, which capture elements that have more than one incoming/outgoing edge. Nodes can also represent *Tasks* and *Events*, which are nodes that have at most one incoming/outgoing edge.

The compliance effect annotations, which represents the fulfillment of a rule on a process element, are captured in Regorous by using a schema called Compliance Check Annotations from which we recall some elements. A *ruleSetList* contains the *ruleSets uri* which is the identification of the rule set. The *conditions* and the *taskEffects* represent the process sequence flow and the tasks respectively (extracted from the BPMN model) and have an associated *effects name* which corresponds to its actual compliance effects annotation.

## 8.2 Pioneering compliance checking in AMASS

The pioneering work on compliance management is conducted in two steps: first a manual exploration, see Section 8.2.1, then a tool-based exploration, see Section 8.2.2.

### 8.2.1 Exploring the usage of defeasible logic and compliance by design.

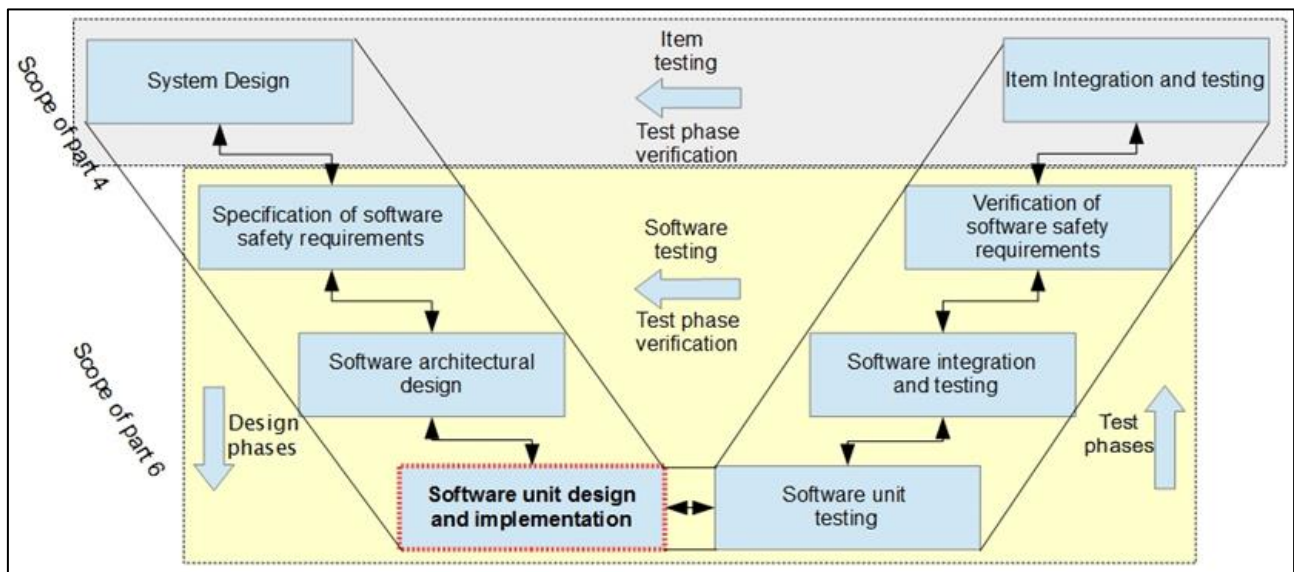
In [88] and in [89] the manual usage of defeasible logic and compliance management was explored.

### 8.2.2 Exploring the usage of REGOROUS for compliance checking

#### 8.2.2.1 Running Example: ISO 26262\*

ISO 26262 [90] addresses functional safety in automotive. ISO 26262 provides a process reference framework based on a V-model, as well as requirements that apply to the activities defined in the process reference model. In Figure 91, the process reference model for product development at the software level is sketched. We focus on the phase *Software unit design and implementation*, described in ISO 26262 part 6, clause 8, and use its requirements to understand the potential of Regorous for compliance checking in the context of safety standards.





**Figure 91.** Product development at the software level

The safety process influences functional safety. Thus, a confirmation review of the safety plan, which includes the compliance checking of the planned process against safety requirements is mandatory. The safety process can be either strictly planned, i.e., including all the activities provided by the reference process, or flexibly planned, i.e., by tailoring activities (omitting/performing them differently). According to ISO 26262:2011 part 2, if a safety activity is tailored, a) the tailoring shall be defined in the safety plan and b) a rationale as to why the tailoring is adequate and sufficient to achieve functional safety shall be available. From a structure perspective, ISO 26262 is divided into parts, which are subdivided into clauses. Some clauses represent phases of the safety process, which also describe activities and tasks.

ISO 26262 uses Automotive Safety Integrity Levels (ASIL), which are levels to specify item's necessary safety requirements. Alternative methods to use in the planning of safety activities (e.g., tables) have to be chosen according to the higher recommendation for the ASIL assigned, but if not, a rationale shall be given that the selected methods comply with the corresponding requirement. Disjoint alternatives are also included in the text of the normative requirements. Frequently recurring expressions, which can guide the reading of the standard, can also be found, e.g., in accordance with. To design a process that is compliant (by design), the following procedure should be followed:

1. We cite parts of the standard related to the activities of the process i.e., the design of the software units, the implementation of the software units and the verification of the software unit design and implementation.
2. We extract the atoms corresponding to the information provided by this phase. An atom may be seen as an important action that is described in the text, e.g. *startingSoftwareUnitDesignAndImplementationPhase*.
3. With the atoms defined in the previous step, and the requirements presented in the standard, we create the FCL rules.
4. The atoms and the rules have to be written in an XML file that has the extension *.ruleset*. In Regorous, the *.ruleset* file is uploaded to enable
  - the user to annotate the task in the process with the atoms, and
  - the compliance checker to analyse the compliance of the tasks with the rules.
5. We prove the rule in small traces of the process to verify compliance by design.

Having the methodology in place, we exemplified its application with a set of standard requirements. First, we cite textual parts of the standard. Figure 92 shows a reformulation of the text given in ISO 26262, Part 6, Clause 8, regarding software units design and implementation.



Objective-1 (if software unit is <u>safety related</u> , as per 8.4.1.)	<u>specify the software units</u> in compliance with the <u>software architectural design</u> and the corresponding <u>software safety requirements</u> .
--	---

**Figure 92.** Example of requirement for ISO 26262-software unit design and implementation

To model the atoms, we select parts of the standard that refer to specific actions or resources that are required. In Figure 92, we have highlighted those specific actions/resources. For example, the title of the phase *Software unit design and implementation* defines the initiation of the phase. For this reason, we create an atom that describes this action: *startingSoftwareDesignAndImplementationPhase*. The total set of atoms for the requirements presented in Figure 92 are:

- *startingSoftwareDesignAndImplementationPhase*
- *specifyingSoftwareUnits*
- *obtainingSoftwareArchitecturalDesign*
- *obtainingSoftwareSafetyRequirements*
- *choosingSoftwareUnit*
- *verifyingASIL*
- *ASILQM*

With the atoms created, and the text of the requirements, we define the FCL rules. For example, a part of the numeral 8.1 says that software units are specified in accordance with the software architectural design. This text can be modelled as FCL rule in the following way:

***r1:startingSoftwareDesignAndImplementationPhase(X) ⇒ [OANPNP] obtainingSoftwareArchitecturalDesign(X)***

This rule means that when starting the software design and implementation phase, the software architectural design must be available. [OANPNP] means that the rule is an O obligation, A achievement, NP non perdurant, and NP non preemptive. In the same way, all the rules corresponding to the requirements of the Figure 92 are modelled:

***r1:startingSoftwareDesignAndImplementationPhase(X) ⇒ [OANPNP] obtainingSoftwareArchitecturalDesign(X)***

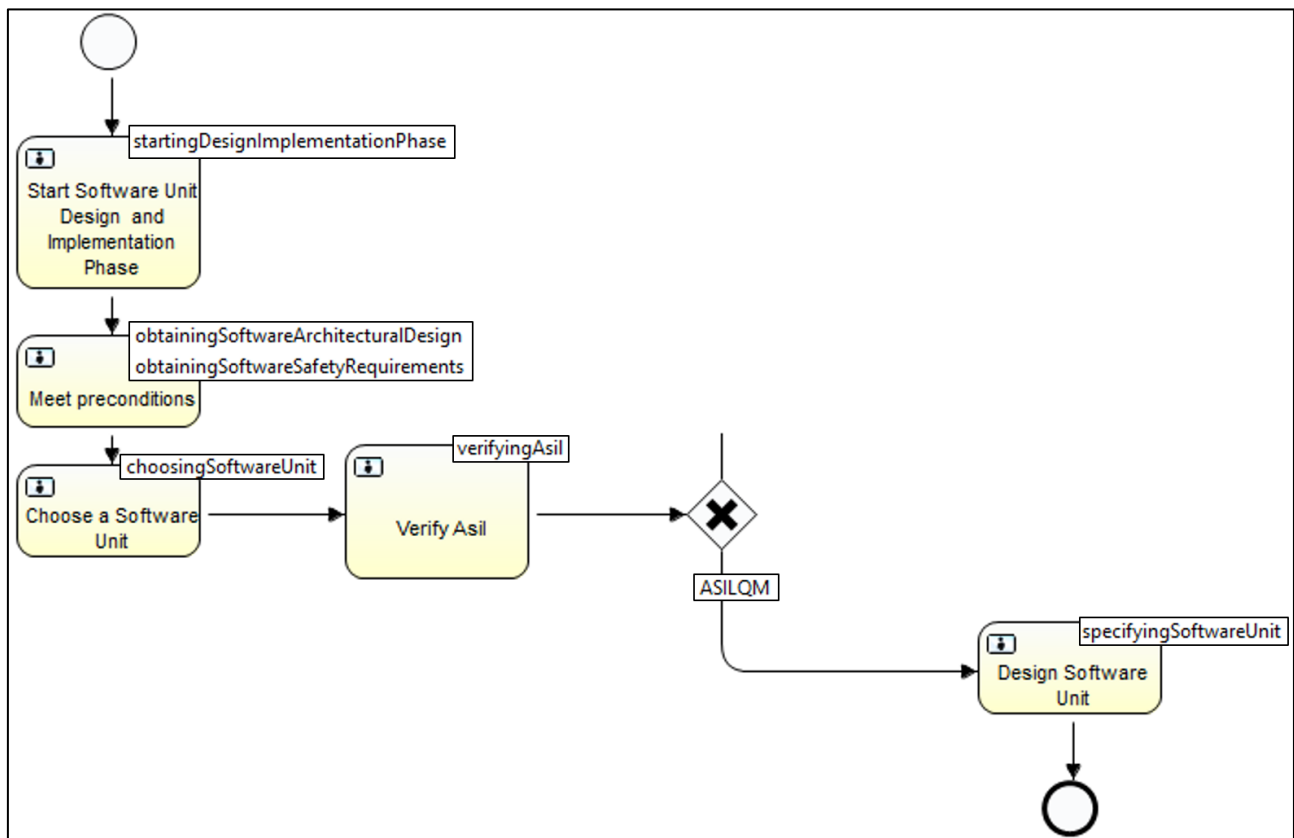
***r2:startingSoftwareDesignAndImplementationPhase(X) ⇒ [OANPNP] obtainingSoftwareSafetyRequirements(X)***

***r3:obtainingSoftwareArchitecturalDesign(X) ⇒ [OANPNP] choosingSoftwareUnit(Y)***

***r4:choosingSoftwareUnit(Y) ⇒ [OANPNP] verifyingAsil(Y)***

***r5:verifyingAsil(Y),ASILQM ⇒ [OANPP] specifyingSoftwareUnits(Y)***

Once the atoms and the rules are written in XML, they are uploaded in Regorous, the process is modelled using BPMN 2.0 and the rules are annotated in the tasks. With all the elements in place, compliance is verified by pressing a button. Figure 93 shows the modelling of the process, an abstraction of the process annotation and the process compliance verification results.



**Figure 93.** Trace 1 of the software unit design and implementation phase

The design of a compliant trace, like the one shown before, was straightforward. However, there were other kinds of requirements that required more analysis.

Set of challenging ISO 26262 requirements are for instance those represented by recommendation tables. The recommendation table regarding the notations to be used for the design of software units, for instance, recommends the usage of four methods, given as four consecutive entries (i.e., all of them should be applied).

As it was mentioned in Section 5.3.2.4, regarding Figure 31, recommendation tables contain variability points, which imply conditionals. The table regarding the notations suggests: Natural language (++ , ++ , ++ , ++), Informal notation (++ , ++ , + , +), semi-formal notation (+ , ++ , ++ , ++), formal notation (+ , + , +).

This kind of requirement is one among many that can be identified by reading the normative clauses in ISO 26262 [82]. From the recommendation table, and the guidelines for interpreting such tables, it is inferred that, for carrying out the design specification of the software units, it is initially necessary to check the ASIL assigned to the software unit. With the ASIL identified, notation(s) according to this ASIL and the highest recommendation level should be preferably selected. However, a rationale should be provided if methods other than those listed are selected. This requirement can be modelled in FCL, by using CDT (Contrary-to-Duty) obligations. The rules, derived from this analysis are:

```

r6:verifyingAsil(Y),¬ASILQM ⇒ [OANPP]choosingCompliantDesignNotation(Y)
r7:choosingCompliantDesignNotation(Y)⇒
    [OAPNP]selectingDesignNotationAccordingASILRecommendation(Y),
    [OAPNP]providingRationaleOnDesignNotationSelected(Y)
  
```

Rule r6 means that once ASIL is verified and it is not QM, a compliant design notation should be chosen. Rule r7 says that a compliant design notation should be selected according to ASIL and recommendations level, but if not, a rationale should be provided. Rule r7 represents a compensation chain for the reparation of a violation (the violation considered here is the non-selection of a recommended notation). This type of rule is called CDT (contrary to duty obligations) in FCL and it suits the modelling of this kind of requirements in ISO 26262. However, the model in Regorous shows that the process that fulfils this kind of rule is not compliant (see Figure

94). The negative result is not a result of a bad practice (or poor understanding) in modelling the rules, but it shows that Regorous is not properly supporting CDTs<sup>19</sup>.

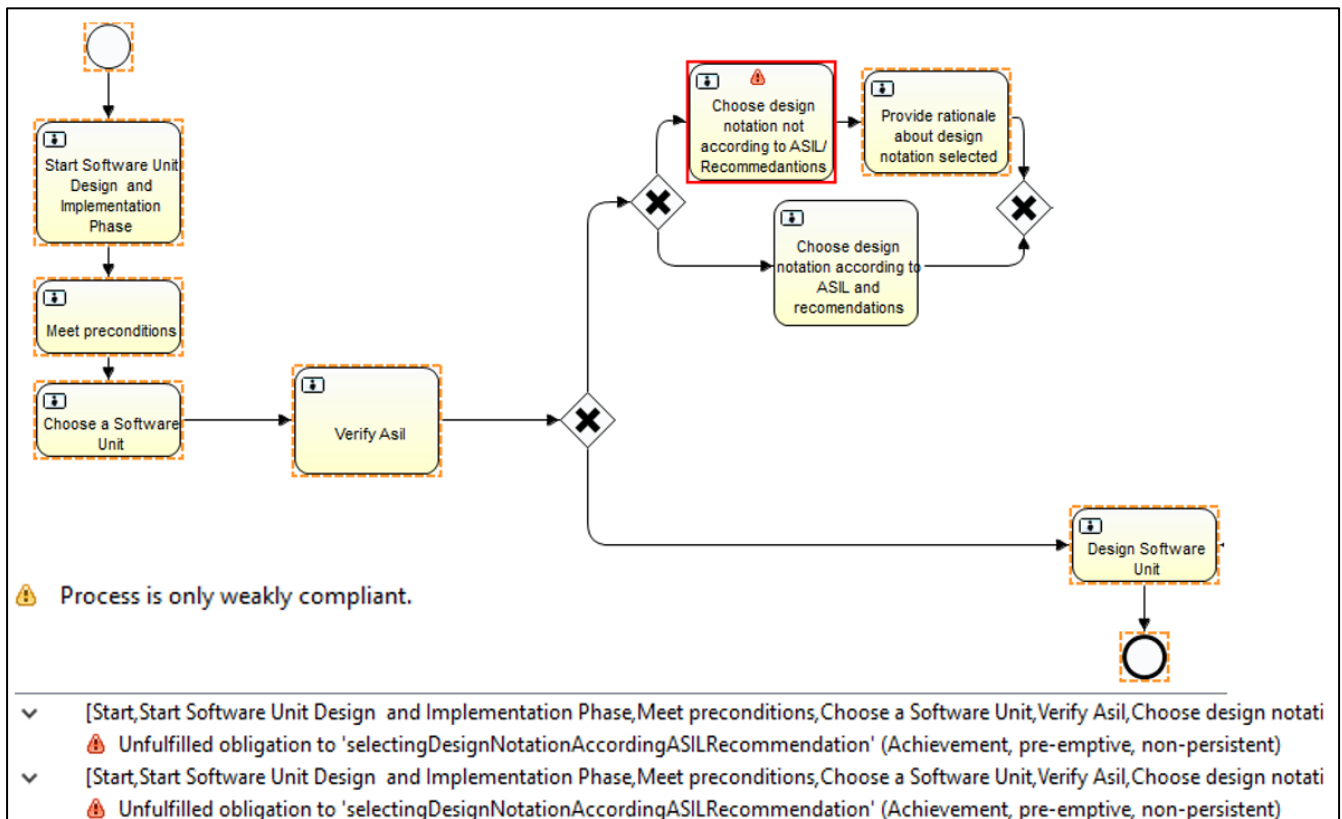


Figure 94. A weakly compliant process checked by Regorous

Rules in FCL can be interpreted and modelled in different ways. Therefore, the rule r7 was re-modelled using a permission. The performed analysis is the following: having a rationale permits to not provide a design notation according to ASIL and recommendation levels. The rules that support this analysis are the following:

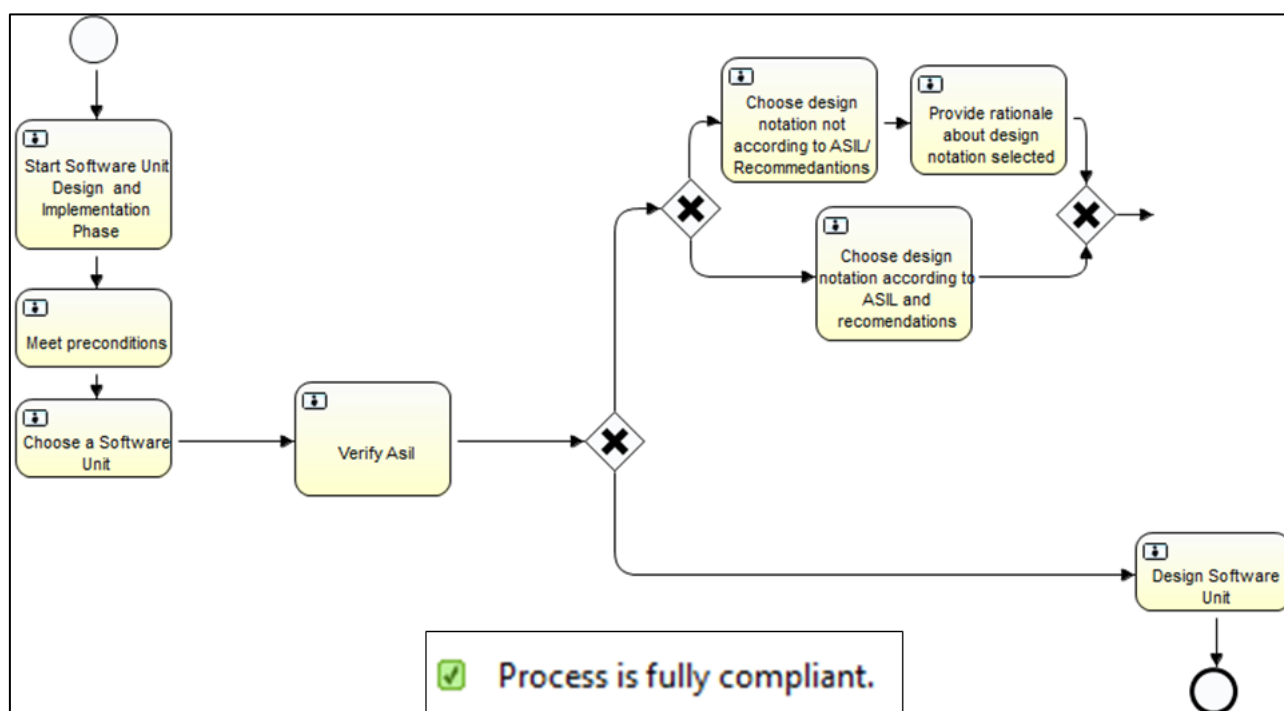
**r7':** *choosingCompliantDesignNotation*(Y) ⇒ [OAPNP]*selectingDesignNotationAccordingASILRecommendation*(Y)

**r8:** *providingRationaleOnDesignNotationSelected*(Y) ⇒ [P]¬*selectingDesignNotationAccordingASILRecommendation*(Y)

**r8>r7**

Rule r7' corresponds to the obligation to provide a design notation according to ASIL and recommendation levels, and rule r8 corresponds to the assumption that the provision of a rationale gives the permission of not selecting a design notation according to ASIL and recommendation levels. Since these two rules are in conflict, it is necessary to add a priority relation, in which the permit is "stronger" than the obligation. The model in Regorous is now compliant (see Figure 95). Modelling the rules in this way is considered sound, since according to legal reasoning and legal theory, permissions can be used to determine that there are not obligations or prohibitions to the contrary (see Section 8.1.2.1).

<sup>19</sup> This assumption was confirmed by the creator of Regorous.



**Figure 95.** Modelling of conditional requirements for ISO 26262: Software unit design and specification

A similar process is applied to all phases. The result of this exercise is the creation of 27 atoms (see Figure 96), 27 rules and 4 superiority relations (see Figure 97), and 20 paths (traces) of compliance (see Figure 98). The complete process, designed to be compliant with ISO 26262 Part 6 Section 8, is presented in Figure 99.

```
<Vocabulary>
  <Term atom="startingDesignImplementationPhase"/>
  <Term atom="specifyingSoftwareUnit"/>
  <Term atom="obtainingSoftwareArchitecturalDesign"/>
  <Term atom="obtainingSoftwareSafetyRequirements"/>
  <Term atom="choosingSoftwareUnit"/>
  <Term atom="verifyingAsil"/>
  <Term atom="ASILQM"/>
  <Term atom="choosingCompliantDesignNotation"/>
  <Term atom="selectingDesignNotationAccordingASILRecommendation"/>
  <Term atom="providingRationaleOnDesignNotationSelected"/>
  <Term atom="choosingCompliantDesignPrinciples"/>
  <Term atom="selectingDesignPrinciplesAccordingASILRecommendation"/>
  <Term atom="providingRationaleOnDesignPrinciplesSelected"/>
  <Term atom="describingFunctionalBehavior"/>
  <Term atom="describingInternalDesign"/>
  <Term atom="implementingSoftwareUnits"/>
  <Term atom="implementingAsModel"/>
  <Term atom="implementingSourceCode"/>
  <Term atom="obtainingModellingGuidelines"/>
  <Term atom="obtainingCodingGuidelines"/>
  <Term atom="achievingImplementationRelatedProperties"/>
  <Term atom="generatingSourceCode"/>
  <Term atom="verifyingSoftwareUnitDesign"/>
  <Term atom="verifyingSoftwareUnitImplementation"/>
  <Term atom="choosingCompliantVerificationMethods"/>
  <Term atom="selectingVerificationMethodAccordingASILRecommendations"/>
  <Term atom="providingRationaleOnVerificationMethods"/>
</Vocabulary>
```

**Figure 96.** Terms required for modelling rules for ISO 26262-Software unit design and Specification





ISO 26262 (ISOAlternative)
r1 - Objective 1: The first objective of this sub-phase is to specify the software units in accordance with the software architectural design
r10 - 8.4.4 Design principles for software unit design and implementation at the source code level as listed in Table 8 shall be applied
r11 - 8.4.4 Design principles for software unit design and implementation at the source code level as listed in Table 8 shall be applied
r12 - 8.4.4 Design principles for software unit design and implementation at the source code level as listed in Table 8 shall be applied
r13 - 8.4.3 The specification of the software units shall describe the functional behaviour and the internal design to the level of detail n
r14 - 8.4.3 The specification of the software units shall describe the functional behaviour and the internal design to the level of detail n
r15 - Objective 2: The second objective of this sub-phase is to implement the software units as specified.
r16 - General: The detailed design will be implemented as a model or directly as source code, in accordance with the modelling or code
r17 - General: The detailed design will be implemented as a model or directly as source code, in accordance with the modelling or code
r18 - General: The implementation-related properties are achievable at the source code level if manual code development is used. If m
r19 - General: The detailed design will be implemented as a model or directly as source code, in accordance with the modelling or code
r2 - Requirement 8.4.1 The requirements of this subclause shall be complied with if the software unit is safety-related.
r20 - General: The detailed design will be implemented as a model or directly as source code, in accordance with the modelling or code
r21 - General: The detailed design will be implemented as a model or directly as source code, in accordance with the modelling or code
r22 - General: The implementation-related properties are achievable at the source code level if manual code development is used. If m
r23 - 8.4.5 The software unit design and implementation shall be verified in accordance with ISO 26262-8:2011 Clause 9, and by applyir
r24 - 8.4.5 The software unit design and implementation shall be verified in accordance with ISO 26262-8:2011 Clause 9, and by applyir
r25 - 8.4.5 The software unit design and implementation shall be verified in accordance with ISO 26262-8:2011 Clause 9, and by applyir
r26 - Objective 3: The third objective of this sub-phase is the static verification of the design of the software units and their implement
r27 - Objective 3: The third objective of this sub-phase is the static verification of the design of the software units and their implement
r3 - Requirement 8.4.1 The requirements of this subclause shall be complied with if the software unit is safety-related.
r4 - Requirement 8.4.1 The requirements of this subclause shall be complied with if the software unit is safety-related.
r5 - General clause: In order to develop a single software unit design both software safety requirements as well as all non-safety-related
r6 - 8.4.2 To ensure that the software unit design captures the information necessary to allow the subsequent development activities to
r7 - 8.4.2 To ensure that the software unit design captures the information necessary to allow the subsequent development activities to
r8 - 8.4.2 To ensure that the software unit design captures the information necessary to allow the subsequent development activities to
r9 - 8.4.4 Design principles for software unit design and implementation at the source code level as listed in Table 8 shall be applied
0 - Rule r8 is superior to rule r7
0 - Rule r11 is superior to rule r10
0 - Rule r19 is superior to rule r16
0 - Rule r25 is superior to rule r24

Figure 97. Rules specification for ISO 26262-Software unit design and specification



Select Execution Paths to Check for Compliance	
<input type="button" value="Select all"/> <input type="button" value="Deselect all"/>	
<input checked="" type="checkbox"/> Path 1	[Start,Start Software Unit Design and Implementation Phase,Meet preconditions,Choose a Software Unit,Verify Asil,Choose design notation according to ASIL and recommendations,Choose Design Principles not according to ASIL/recommendations,Provide rationale about design principles selected,Design Software Unit,Describe functional behavior,Describe internal Design,Choose implementation type,Implement the software units as a model,Obtain modelling guidelines,Generate source code,Checking implementation-related properties,Choose compliant verification methods,Choose verification Methods not according to ASIL/recommendations,Provide rationale about verification methods selected,Verify Software Unit design,Verify Software Unit Implementation,End]
<input checked="" type="checkbox"/> Path 2	[Start,Start Software Unit Design and Implementation Phase,Meet preconditions,Choose a Software Unit,Verify Asil,Choose design notation according to ASIL and recommendations,Choose Design Principles not according to ASIL/recommendations,Provide rationale about design principles selected,Design Software Unit,Describe functional behavior,Describe internal Design,Choose implementation type,Implement the software units as a model,Obtain modelling guidelines,Generate source code,Checking implementation-related properties,Choose compliant verification methods,Choose verification methods according to ASIL/recommendations,Verify Software Unit design,Verify Software Unit Implementation,End]
<input checked="" type="checkbox"/> Path 3	[Start,Start Software Unit Design and Implementation Phase,Meet preconditions,Choose a Software Unit,Verify Asil,Choose design notation according to ASIL and recommendations,Choose Design Principles not according to ASIL/recommendations,Provide rationale about design principles selected,Design Software Unit,Describe functional behavior,Describe internal Design,Choose implementation type,Implement the software units as code directly,Obtain Coding guidelines,Checking implementation-related properties,Choose compliant verification methods,Choose verification Methods not according to ASIL/recommendations,Provide rationale about verification methods selected,Verify Software Unit design,Verify Software Unit Implementation,End]
<input checked="" type="checkbox"/> Path 4	[Start,Start Software Unit Design and Implementation Phase,Meet preconditions,Choose a Software Unit,Verify Asil,Choose design notation according to ASIL and recommendations,Choose Design Principles not according to ASIL/recommendations,Provide rationale about design principles selected,Design Software Unit,Describe functional behavior,Describe internal Design,Choose implementation type,Implement the software units as code directly,Obtain Coding guidelines,Checking implementation-related properties,Choose compliant verification methods,Choose verification methods according to ASIL/recommendations,Verify Software Unit design,Verify Software Unit Implementation,End]
<input checked="" type="checkbox"/> Path 5	[Start,Start Software Unit Design and Implementation Phase,Meet preconditions,Choose a Software Unit,Verify Asil,Choose design notation according to ASIL and recommendations,Choose design principles according to ASIL/Recommendations,Design Software Unit,Describe functional behavior,Describe internal Design,Choose implementation type,Implement the software units as a model,Obtain modelling guidelines,Generate source code,Checking implementation-related properties,Choose compliant verification methods,Choose verification Methods not according to ASIL/recommendations,Provide rationale about verification methods selected,Verify Software Unit design,Verify Software Unit Implementation,End]
<input checked="" type="checkbox"/> Path 6	[Start,Start Software Unit Design and Implementation Phase,Meet preconditions,Choose a Software Unit,Verify Asil,Choose design notation according to ASIL and recommendations,Choose design principles according to ASIL/Recommendations,Design Software Unit,Describe functional behavior,Describe internal Design,Choose implementation type,Implement the software units as a model,Obtain modelling guidelines,Generate source code,Checking implementation-related properties,Choose compliant verification methods,Choose verification methods according to ASIL/recommendations,Verify Software Unit design,Verify Software Unit Implementation,End]
<input checked="" type="checkbox"/> Path 7	[Start,Start Software Unit Design and Implementation Phase,Meet preconditions,Choose a Software Unit,Verify Asil,Choose design notation according to ASIL and recommendations,Choose design principles according to ASIL/Recommendations,Design Software Unit,Describe functional behavior,Describe internal Design,Choose implementation type,Implement the software units as code directly,Obtain Coding guidelines,Checking implementation-related properties,Choose compliant verification methods,Choose verification Methods not according to ASIL/recommendations,Provide rationale about verification methods selected,Verify Software Unit design,Verify Software Unit Implementation,End]

**Figure 98.** Definition of the traces for the process Software unit design and specification

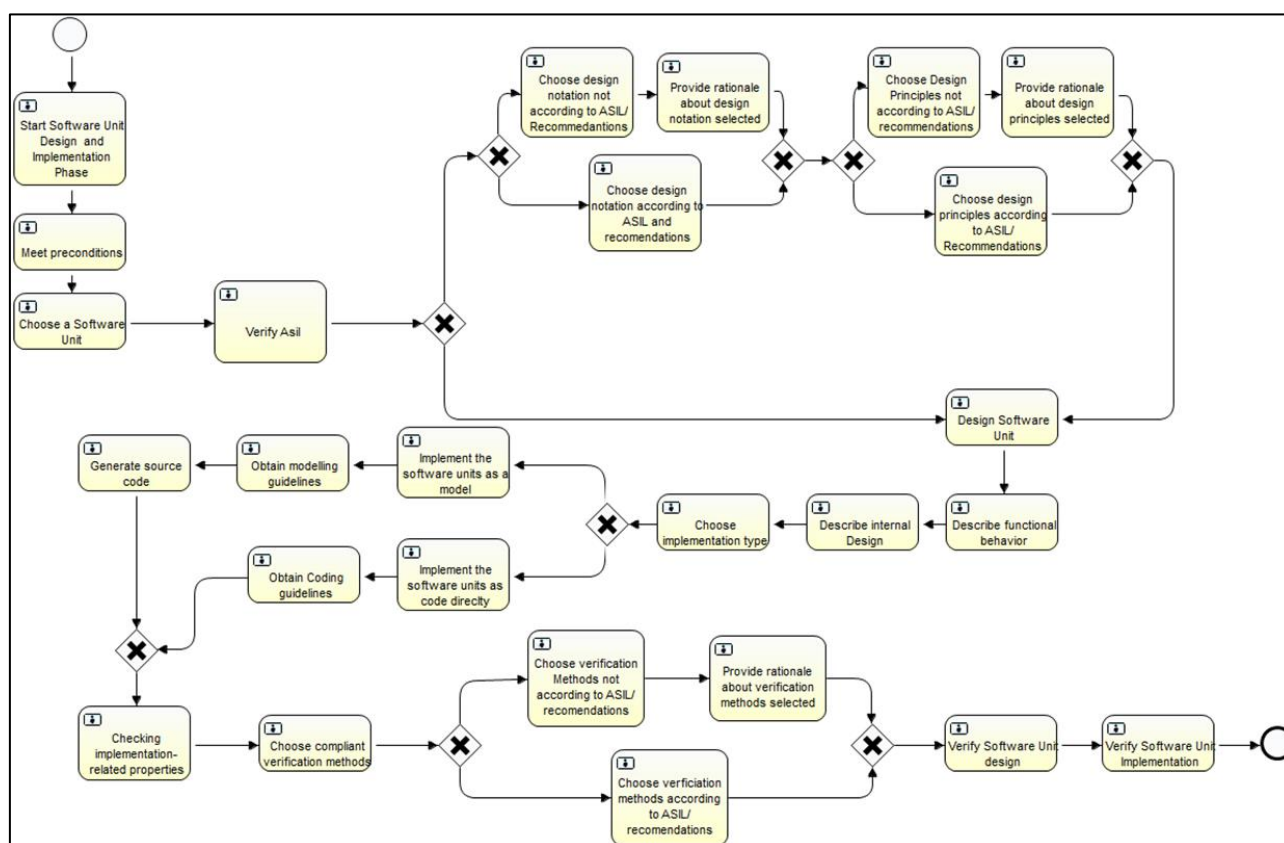


Figure 99. Complete model of ISO 26262-Software unit design and specification

### 8.2.3 Exploration conclusions

Regorous's capabilities have been experimented and considered of important value, even though more explorations are required to measure its usefulness for modelling compliant processes for engineering cyber-physical systems. However, during the modelling of the compliant process, some difficulties are present:

1. Rules had to be modelled "by hand" in a text processor, like notepad. An FCL editor was not provided with the Regorous Process Designer. Therefore, the process of creation of the rules is error prone, and time-consuming.
2. The notation presented for the rule in FCL, i.e., [OAPP], which means Obligation, Achievement, Preemptive, Perdurant according to the theoretical background presented in Section 8.1.2.2, is not implemented in the exact same way in Regorous (Obligation, Achievement, Perdurant, Preemptive). There is a gap between the theoretical foundations and the tool implementation.
3. The use of CDT (Contrary-to-Duty Obligations) could be beneficial for the definition of the rule in safety standards. However, the version of FCL used in Regorous does not support CDT, in the way described in Section 8.1.2.1. This means that Regorous is not able to discard a rule that is violated (and it is not perdurant) but being compensated by another rule. In the exercise presented in Section 8.2.2, permissions were used to overcome this problem, meaning that if there is a permission to do something, there is not an obligation to do something else. This is a way to compensate the violation of an obligation, and therefore to carry out requirements that have conditional nature.
4. The current version of Regorous allows the modelling of process tasks, but not other types of process elements, like roles, work products, guidelines, etc.



### 8.3 Conceptual solution

In the context of AMASS, SPEM2.0/EPF Composer has been selected to model those process elements that represent plans. This selection is based on the fact that SPEM 2.0 is well suited for modelling software process, not only for the provision of generic process concepts (e.g., activity, work products), but also for the extension mechanisms for modelling and documenting a wide range of processes [91]. Besides, SPEM 2.0 is a good candidate to model a process mandated by safety standards [92] and to some extent, it also support the creation of compliance tables, i.e., the mapping between standards requirements and process elements [96]. Therefore, this tool is also recommended to be used during the process of compliance checking.

However, SPEM2.0/EPF Composer does not provide the compliance checking capability needed for the AMASS purposes. Based on what was recalled in Sections 8.1 and 8.2, it becomes evident that FCL is a language that can be used for providing the process compliance checking capability, currently missing in SPEM2.0/EPF Composer. Moreover, Regorous, which is the compliance checker that supports reasoning with FCL, can be considered as part of the tool support to enrich software process modelling with SPEM2.0/EPF Composer. An enriched SPEM2.0/EPF Composer would be able to offer the means required for modelling process plans and its subsequent compliance checking. In particular, it should be possible to model FCL rules in a rule editor, will support the creation of the FCL rules is an easier and effective way. FCL rules will be used to annotate process elements provided by SPEM2.0/EPF Composer that will be checked with Regorous. As a consequence, Regorous will provide a compliance report from which it would be easier to analyse and improve software process compliance.

The current version of Regorous clearly supports the compliance checking of tasks in a process. However, the support of other process elements is less clear. Therefore, it is likely that new analysis is needed to increase compliance capabilities to Regorous.



## 9. Potential design solutions for the compliance checking

In this section, design solutions for compliance checking vision are sketched. In Section 9.1, architectural solutions in which Regorous process designer plays an important role are given. In Section 9.2, the architecture of a process compliance checker, created from scratch using FCL, and the defeasible logic reasoner SPINdle are presented.

### 9.1 Proposals for adapting Regorous to the needs of AMASS

In this section, potential proposals for modification of the Regorous architecture (presented in Figure 88) for its adaptation to the AMASS needs are proposed. Before explaining those proposals, for sake of clarity, it is worth to recall that, as presented in [73], Regorous is process modelling language agnostic, which means that even if the current implementation is based on BPMN, all what Regorous needs is to get a description of the process and the annotation of each task with the FCL rules.

The first proposal, depicted in Figure 100, proposes the addition of an editor package, which contains two elements: *EPF Composer*, for facilitating reuse of process elements, and a *Rule Editor*, for facilitating the creation of rule sets.

EPF Composer is expected to provide, as an output, the process models that have to be translated (via model transformation) to Eclipse Activiti BPMN 2.0. The rule editor will provide the ruleset file, which is currently needed by Regorous. Regorous is expected to maintain its actual functionality, providing compliance checking of process tasks without the implementation of the reparation chains. In addition, the tasks of the model provided via SPEM2.0/EPF Composer have to be annotated in BPMN 2.0.

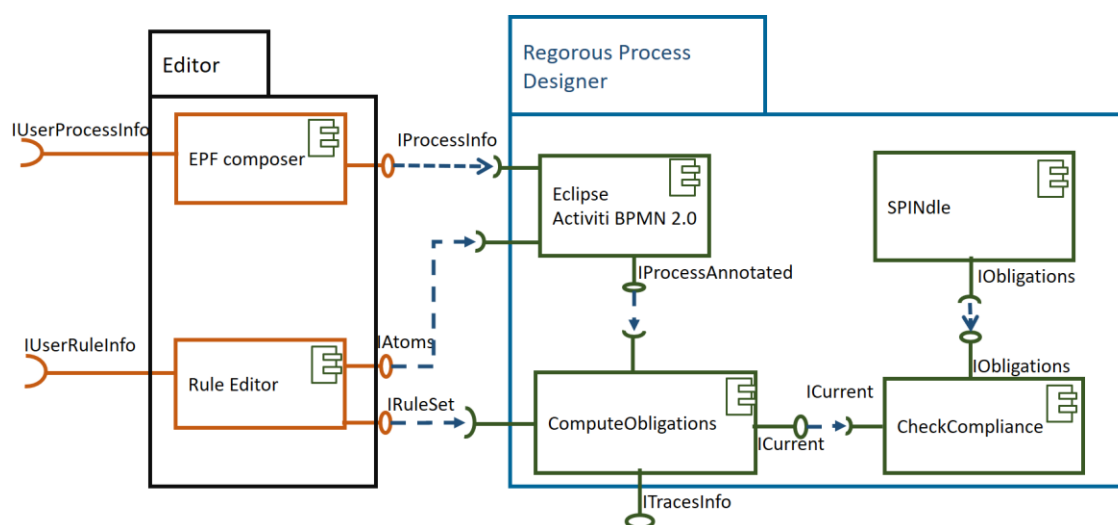
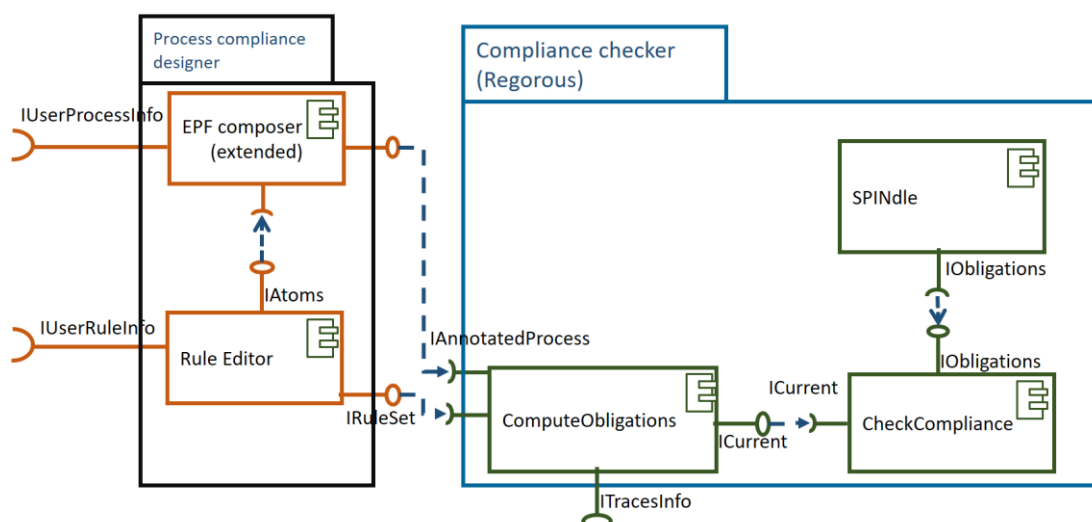


Figure 100. Adding EPF Composer + rule editor

By implementing the solution depicted in Figure 100, the usual capabilities provided by Regorous remain. In addition, the rule editor is expected to facilitate the creation of the rule sets required by Regorous. The addition of EPF Composer serves two purposes. The first purpose is the generation of the bridge that is required between AMASS core structure and Regorous tool. The second purpose is the provision of process elements beyond tasks, a characteristic not provided by Regorous.

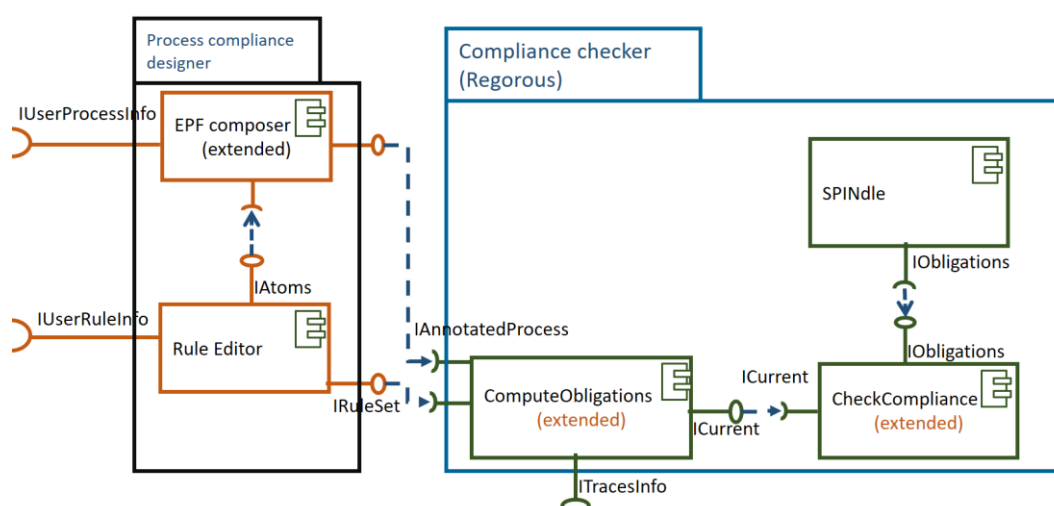
The second proposal, depicted in Figure 101, requires, in addition to the first proposal, an extended version of EPF Composer, to provide the user the possibility to add semantic annotations to the process elements, directly in the model created in EPF Composer. The idea with the provision of this extended version of EPF Composer is to replace Eclipse Activiti BPMN 2.0 to avoid the translation from an UMA-compliant process model to the BPMN-compliant process model, and to add process elements beyond tasks to the process model. However,

the compliance analysis is expected to be maintained only for the process tasks, since the provision of compliance to the other process elements require a modification of the *ComputeObligations* algorithm. Reparation chains are not added since this functionality is presented in Regorous, whose algorithms are not planned to be modified in this version of the solution.



**Figure 101.** Replacing Activiti BPMN 2.0 with EPF Composer + rule editor

These two first proposals have the advantages that the underlying methodology used in Regorous process designer will facilitate and speed up the implementation of a compliance tool checker that partially covers AMASS requirements. However, Regorous is a proprietary software, and licenses are required for its utilization. In addition, Regorous has been implemented in an older version of Eclipse, which requires to be updated. Changes in the software are only permitted for Regorous owners. Thus, changes cannot be done in the context of the AMASS project, unless policies about licenses for use and modification of the software of Regorous are agreed with Regorous owners (previously called Nicta<sup>20</sup>). In the light of agreements with the Regorous owners, a third proposal can be designed (see Figure 102).



**Figure 102.** Replacing BPMN with EPF Composer + rule editor + extended ComputeObligations/CheckCompliance components

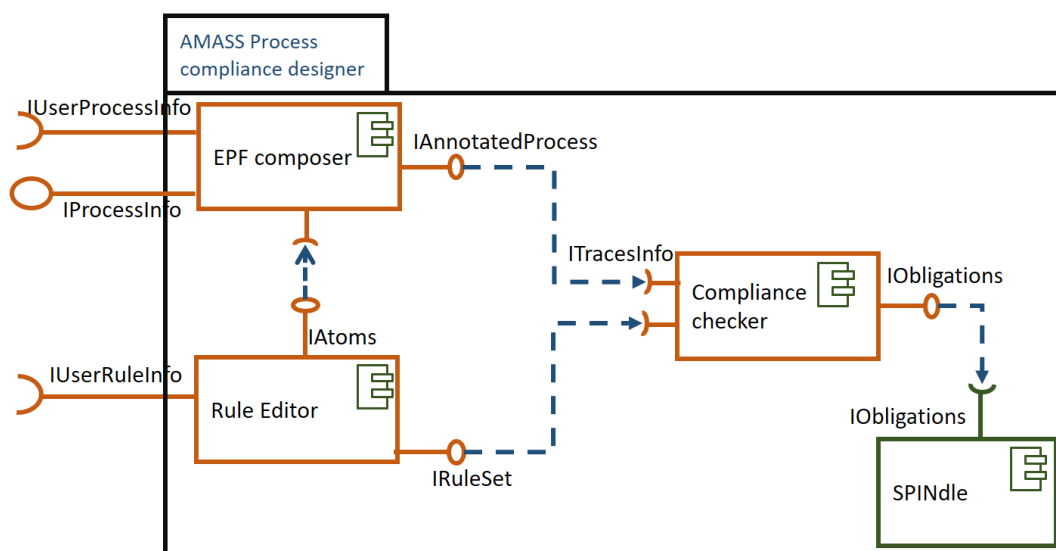
<sup>20</sup> <https://www.data61.csiro.au/>



The solution presented in Figure 102 adds to the second proposal (presented in Figure 101) an extended functionality for the algorithms *ComputeObligations* and *CheckCompliance* (which belong to the Regorous tool). The extended functionalities are expected to provide compliance checking for process elements beyond tasks, and to include CDT obligations (reparation chains) to formulate the rules. This proposal will cover all the requirements for compliance checking in the context of safety standards, but the tool implemented may be subject to vendor license fees, and the provided code may not be open source.

## 9.2 Creating an AMASS process compliance checker from scratch

A completely new compliance checker can be designed, using FCL (presented in Section 8.1.2), the methodology for compliance by design used in Regorous (presented in section 8.1.3 ), and the defeasible logic reasoner (SPINdle). As presented in Section 8.1.3.3, SPINdle is an open source application, which can be used in the design and implementation of a completely new tool for Process compliance in the context of AMASS (see Figure 103).



**Figure 103.** AMASS Process Compliance Designer

In this version, a compliance checker, that supports FCL rules, the computation of obligations for all process elements, and the check of compliance including reparation chains should be designed. The compliance checker functionality will take as input the annotated process from the EPF composer and the rule set provided by the rule editor. The functionality provided by the Compliance checker should allow:

- Computing of obligations for all process elements, not only tasks.
- Compliance checking for all the traces in the process.
- Including reparation chains as an alternative for modelling rules in FCL.

## 9.3 Pros and cons of the architectural design solutions

Table 15 summarizes the pros and cons of the architectural design solutions.

**Table 15.** Pros and cons of the architectural design solutions

Solution	Pros	Cons
Adding EPF Composer + rule editor	<ul style="list-style-type: none"><li>• Short term implementation time.</li><li>• Provide compliance analysis of process tasks.</li><li>• Addition of EPF Composer (AMASS selected tool) as graphical interface for the process model.</li></ul>	<ul style="list-style-type: none"><li>• Implemented tool is under vendor/proprietary licenses.</li><li>• Process elements different that tasks cannot be analysed for compliance.</li><li>• Contrary-to-duty obligations or reparation chains are not implemented.</li><li>• Requires a process of model-to-model transformation to import the process model from EPF composer to BPMN 2.0.</li></ul>
Replacing BPMN by an extended version of EPF Composer + rule editor	<ul style="list-style-type: none"><li>• Short term implementation time.</li><li>• EPF Composer (AMASS selected tool) will be used as the direct process modeller. Therefore model-to-model transformations are not required, increasing response time of the tool.</li></ul>	<ul style="list-style-type: none"><li>• Implemented tool is under vendor/proprietary licenses.</li><li>• Process elements different that tasks cannot be analysed for compliance.</li><li>• Contrary-to-duty obligations or reparation chains are not implemented.</li></ul>
EPF Composer + rule editor + extended Compute obligations algorithm + extended Check compliance algorithm	<ul style="list-style-type: none"><li>• Medium term implementation time.</li><li>• EPF composer (AMASS selected tool) will be used as the direct process modeller. Therefore, model to model transformations are not required, increasing response time of the tool.</li><li>• Compliance checking for other process elements beyond tasks.</li><li>• Contrary to duty obligations or reparation chains included in the tool.</li></ul>	<ul style="list-style-type: none"><li>• Implemented tool is under vendor/proprietary licenses.</li></ul>
Creation a new compliance checker	<ul style="list-style-type: none"><li>• EPF Composer (AMASS selected tool) will be used as the direct process modeller. Compliance checking for other process elements beyond tasks.</li><li>• Contrary to duty obligations or reparation chains included in the tool.</li><li>• Open source tool.</li></ul>	<ul style="list-style-type: none"><li>• Long term implementation, probably beyond the limits of AMASS project.</li></ul>



## 10. AMASS design solution for compliance checking (\*)

In this chapter, the AMASS design for the compliance checking vision (presented in Chapter 8-9) is detailed. The chapter is structured as follows: in 10.1, safety compliance patterns are explained. In Section 10.2, the mechanisms to model SPEM 2.0-compatible process models for compliance checking are described. Finally, in Section 10.3, the mechanism to generate Regorous inputs are described.

### 10.1 Safety compliance patterns

Formalizing safety requirements in FCL, recalled in Section 8.1.2, requires skills, which cannot be taken for granted. Patterns, which are “*abstractions from concrete forms which keep recurring in specific non-arbitrary context*” [93], could represent a solution. For this reason, in [94], safety compliance patterns are defined, by following Dwyer’s et. al’s property specification pattern style, recalled in Section 8.1.4. In this section, a definition of safety compliance patterns is presented, as well as a set of ISO 26262-specific FCL compliance patterns to facilitate rules formalization.

#### 10.1.1 Safety compliance patterns

Automatic compliance checking of process, as presented in Figure 87, involves the definition of a finite state model of the process, where normative safety requirements provide the permissible states of the process elements. In this sense, a process can be verified in a similar way as is done with a system. Thus, the specification pattern definition, presented in Section 8.1.4, can be translated into the safety process compliance checking context as follows: *safety compliance patterns are patterns that describe commonly occurring normative safety requirements on the permissible state sequence of a finite state process model*. With this definition, it is possible to develop a mapping between specification patterns and safety compliance patterns, as follows: the presence of a state in a system can be interpreted as the state of the obligation imposed to an element in the process, and the scope corresponds to the interval in a process when the obligations formulated by the pattern are in force.

#### 10.1.2 ISO 26262-related compliance patterns identification

For identifying a safety compliance pattern in ISO 26262, five methodological steps have been delineated, as follows:

1. The first step consists of the selection of a recurring structure in the standard since safety requirements in ISO 26262 have implicit and explicit structures.
2. The second step is the description of the obligation for compliance, namely, the reasons why the structure is required for safety compliance.
3. The third step is the pattern description, based on similar (or a combination of) behaviors of the patterns described by Dwyer et al.’s (See Section 8.1.4) This description is contextualized to safety compliance, based on the mapping presented in Section 10.1.1. In this step, a name, which reflects the related obligation for compliance, is assigned to the safety pattern.
4. The fourth step is the definition of the scope of the pattern, which is also based on Dwyer et al.’s work.
5. The fifth step is the formalization in FCL. To formalize the pattern, the scope defined for the pattern requires being mapped into the rule notations provided by FCL. Therefore, a global scope, which represents the entire process model execution, can be mapped to maintenance obligation, which represents that an obligation has to be obeyed during all instants of the process interval. A before scope, which includes the execution of the process model up to a given state, can be mapped to the concept of preemptive obligation, which represents that an obligation could be fulfilled even before it is in force. An after scope, which includes the execution of the process model until a given state, can be mapped to the concept non-preemptive obligation, which represents that an obligation cannot be fulfilled until it is in force. Table 16 presents the mapping previously described.

**Table 16.** Mapping of the patterns scope into FCL rule notation

Dwyer's et al's patterns scope	FCL rule notation
Global	Maintenance Obligation (OM)
Before	Preemptive Obligation (OAP)
After	Non-preemptive Obligation (OANP)

It should be noted that, in safety compliance, it is possible to define exceptions for the rules. Therefore, if the obligation admits an exception, the part of the pattern that corresponds to the exception is described as a permission, since, as recalled in Section 8.1.2, if something is permitted the obligation to the contrary does not hold. The obligation, to which the exception applies, is modeled as non-perdurant, since the permission is not a violation of the obligation, and therefore the obligation does not persist after the permission is granted. In this case, the obligation and a permission have contradictory conclusions, but the permission is superior since it represents an exception.

### 10.1.3 ISO 26262-related compliance patterns definition

The methodological steps, presented in Section 10.1.2, are used to define an initial set of four ISO 26262 - related FCL compliance patterns.

1. **Pattern:** Address Phase. **Recurring structure:** A phase. **Obligation for compliance:** Every phase proposed by the safety model must be addressed. A phase can be omitted if tailoring is performed and a rationale is provided. **Pattern description:** Universality + absence - A phase must occur. Not addressing the phase requires its tailoring and the provision of a rationale. **Scope:** Global. **FCL mapping:** A maintenance obligation address{Phase} is triggered by a previous task {optionalTriggeringObligation}, which can be empty if the phase is checked for compliance in isolation from the other phases. The permission for not address{phase} requires two antecedents, tailor{Phase} and rationaleForOmitting{Phase}

$$\begin{aligned}
 r: \{ \text{optionalTriggeringObligation} \} &\Rightarrow [\text{OM}] \text{address}\{\text{Phase}\} \\
 r': \text{tailor}\{\text{Phase}\}, \text{rationaleForOmitting}\{\text{Phase}\} &\Rightarrow [\text{P}] - \text{address}\{\text{Phase}\} \\
 r' &> r
 \end{aligned}$$

2. **Pattern:** Perform Preconditions. **Structure:** The structure implicit in the expression in accordance with. **Obligation for compliance:** A task is prohibited until the preconditions are performed. **Pattern description:** Absence + precedence - A given task cannot occur within a scope. The task is permitted to be performed if the preconditions are performed. **Scope:** After. **FCL mapping:** A rule triggered by a previous rule {TriggeringObligation} prohibits the performing of the task perform{Task}. The permission for not perform{Task} is granted after the preconditions are fulfilled perform{Preconditions}.

$$\begin{aligned}
 r: \{ \text{TriggeringObligation} \} &\Rightarrow [\text{OANONP}] - \text{perform}\{\text{Task}\} \\
 r': \text{perform}\{\text{Precondition1}\}, \dots, \text{perform}\{\text{PreconditionN}\} &\Rightarrow [\text{P}] \text{perform}\{\text{Task}\} \\
 r' &> r
 \end{aligned}$$

3. **Pattern:** Select Disjoint Methods. **Structure:** Structure implicit when the word or is used to list two methods. **Obligation for compliance:** Only one method can be selected from a list of two. **Pattern description:** Existence + absence - A given method can be selected within a scope. The presence of a second method derogates the selection of the first method. **Scope:** After. **FCL mapping:** A rule triggered by previous obligations {TriggeringObligation} imposes the obligation of selecting a method select{Method}. The selection of a second method select{Method2}, derogates the previous selection select{Method1}.

$$\begin{aligned}
 r: \{ \text{TriggeringObligation} \} &\Rightarrow [\text{OANONP}] \text{select}\{\text{Method1}\} \\
 r': \text{select}\{\text{Method2}\} &\Rightarrow [\text{P}] - \text{select}\{\text{Method1}\} \\
 r' &> r
 \end{aligned}$$

4. **Pattern:** Select alternative methods. **Structure:** Alternative methods given in tables. **Obligation for compliance:** Methods should be selected according to ASIL/recommendation levels. Alternative methods can be selected if a rationale is provided. **Pattern description:** Response + absence - A given obligation has to occur. The provision of a rationale grants the permission to derogate the obligation. **Scope:** After. **FCL mapping:** A rule triggered by previous obligations {TriggeringObligation} imposes the selection of methods according to the requirements select{mandatoryMethods}. The provision of the rationale is the permission that derogates the obligation.

$$\begin{aligned} r: \{ \text{TriggeringObligation} \} &\Rightarrow [\text{OANONP}] \text{select}\{\text{mantdatoryMethods}\} \\ r': \text{provideRationaleForNotSelect}\{\text{mandatoryMethod}\} &\Rightarrow [\text{P}] - \text{select}\{\text{mandatoryMethod}\} \\ r' &> r \end{aligned}$$

#### 10.1.4 ISO 26262-related compliance patterns instantiation

In this section, the patterns are instantiated. First, the definition of the (sub-)phase Software Unit Design and Implementation, which is presented in Figure 104, can be formally represented by instantiating the pattern *AddressPhase*. In the following way:

$$\begin{aligned} r_1 &\Rightarrow [\text{OM}] \text{address}\{\text{SwUnitDesignAndImplementation}\} \\ r_1': \text{tailor}\{\text{SwUnitDesignAndImplementation}\}, \text{rationaleForOmiting}\{\text{SwUnitDesignAndImplementation}\} \\ &\Rightarrow [\text{P}] - \text{address}\{\text{SwUnitDesignAndImplementation}\} \\ r_1' &> r_1 \end{aligned}$$

Objective-1 (sub-phase: software unit design and implementation)	<u>specify the software units in compliance with the software architectural design and the corresponding software safety requirements.</u>
---	--

**Figure 104.** Requirement that represents the initiation of the software unit design and implementation (sub-)phase

The first objective of the (sub-)phase, also presented in Figure 104, has the expression *in compliance with*, which can be represented with the pattern *Perform Preconditions*. Specifically, the software architectural design and the associated safety requirements are preconditions to specify the software units. It can be assumed that the triggering rule is addressSwUnitDesignAndImplementationEnabling.

$$\begin{aligned} r_2: \{ \text{addressSwUnitDesignAndImplementation} \} &\Rightarrow [\text{OANONP}] - \text{perform}\{\text{SpecifySwUnits}\} \\ r_2': \text{perform}\{\text{ProvideSoftwareArchitecturalDesign}\}, \text{perform}\{\text{ProvideSafetyRequirments}\} \\ &\Rightarrow [\text{P}] \text{perform}\{\text{SpecifySwUnits}\} \\ r_2' &> r_2 \end{aligned}$$

Figure 105 represents a requirement, which mentions the use of two disjoint implementation strategies, namely implementation as a model or directly as source code. Therefore, this requirement can be modeled using the pattern *Select Disjoint Methods*. It can be assumed that the triggering rule is implementingSwUnit.

$$\begin{aligned} r_3: \{ \text{implementingSwUnit} \} &\Rightarrow [\text{OANONP}] \text{select}\{\text{ImplementingAsSourceCode}\} \\ r_3': \text{select}\{\text{ImplementingAsModel}\} &\Rightarrow [\text{P}] - \text{select}\{\text{ImplementingAsSourceCode}\} \\ r_3' &> r_3 \end{aligned}$$

8.2	Detailed design will be given as a model (in compliance with the modelling guidelines) or as source code (in compliance with the coding guidelines)
-----	---

**Figure 105.** Requirement that represents the selection of disjoint implementation strategies

Finally, Figure 106 represents a requirement in which mandatory notations should be selected. This requirement can be represented by using the pattern *Select alternative methods*. It is assumed that the triggering rule is performSpecifySwUnit.

$$\begin{aligned}
 r_4: \{SpecifySwUnits\} &\Rightarrow [OANONP]select\{mandatoryNotationsForSwDesign\} \\
 r_4': provideRationaleForNotSelect\{mandatoryNotationsForSwDesign\} \\
 &\Rightarrow [P] - select\{mandatoryNotationsForSwDesign\} \\
 r_4' &> r_4
 \end{aligned}$$

**8.4.2** To ensure that the software unit design captures the information necessary to allow the subsequent development activities to be performed correctly and effectively, the software unit design shall be described using the notations listed in Table 7.

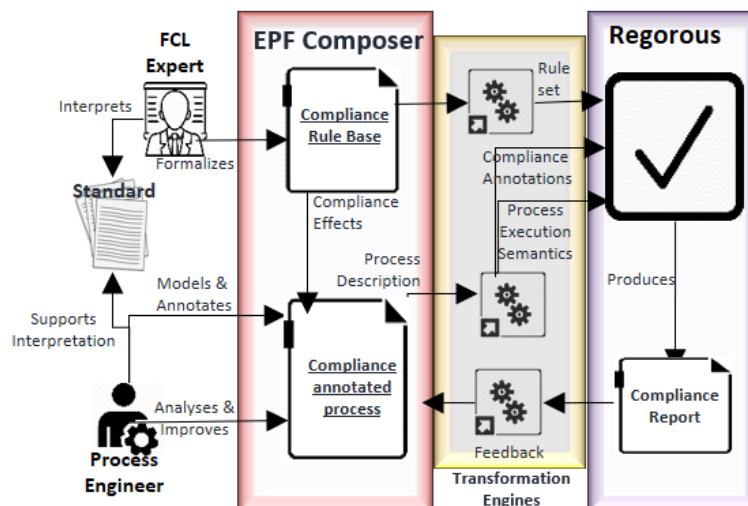
**Table 7 — Notations for software unit design**

Methods		ASIL			
		A	B	C	D
1a	Natural language	++	++	++	++
1b	Informal notations	++	++	+	+
1c	Semi-formal notations	+	++	++	++
1d	Formal notations	+	+	+	+

**Figure 106.** Requirement that represents the selection of mandatory notations

## 10.2 Modelling SPEM 2.0-compatible process models for compliance checking

This section explains the first steps for concretizing the AMASS solution for compliance checking in which SPEM 2.0-compatible process models can be used for compliance checking with Regorous. Recalling, this solution consists of the combination of process modeling capabilities via SPEM 2.0 [6] (Systems & Software Process Engineering Metamodel) reference implementation, specifically by using EPF (Eclipse Process Framework) Composer, and compliance checking capabilities via Regorous [8], an FCL-based reasoning methodology and tool, used in the business context.



**Figure 107.** AMASS Compliance Checking Vision

Figure 107 depicts the graphical representation of the AMASS Compliance Checking Vision in which two roles are required, namely, the *process engineer*, who should support the interpretation of the standard's requirements, model, annotate the process, and analyze the compliance report; and an *FCL expert*, who should interpret standard's requirements and formalize them in FCL. The figure is divided in three areas, which represent the tool support required, as follows:





1. Region 1 (surrounded by a red line) delimitates the tool support required for describing rule sets and for modelling and annotating software processes with compliance effects.
2. Region 2 (surrounded by a purple line) describes Regorous compliance checking.
3. Region 3 (surrounded by a yellow line) delimitates the *mechanisms to ensure SPEM 2.0 and Regorous compatibility*.

### 10.2.1 Mechanisms to annotate software process models

This step represents the mechanism that should be implemented in the Region 1 (surrounded by a red line in Figure 107) to represent rule sets and model and annotate (with compliance effects) the software processes. *Compliance effects* are those effects that emerge from the cumulative interactions between the process tasks, producing the desired global properties mandated by the regulations. Technically, compliance effects are extracted from the set of formulas of the logic, i.e., compliance effects correspond to the premises and conclusions that compound the rules. SPEM 2.0 and the open source implementation, called EPF composer, offers sufficient elements to describe the software process required by Regorous. In particular, SPEM 2.0 defines reusable content method elements for representing a variety of process models, including elements to support textual descriptions in a variety of ways, called *Guidance types*. Two different guidance kinds are of interest, namely the *Reusable Asset*, in which rule set information can be captured and the *Concept* in which key information regarding compliance effects can be recorded. These two guidance kinds have the particularity that they can be added, or in other words, they can be annotated to the process models. To accommodate the process descriptions required by Regorous, three plugins, as done by the IBM approach for mapping standards requirements presented in [96], are adopted in the following way:

1. **Plugin for capturing standard's requirements:** In the method authoring of EPF composer, it is captured the standard's requirements using custom categories. The root of the custom categories is the name of the standard. The novelty added to this plugin is that the rule set and the rules are added by using a customized reusable asset for the former and customized concepts for the latter. The rules are associated to the corresponding standard's requirements.
2. **Plugin for capturing process elements:** In the method authoring of EPF composer, the process elements required to support the software process modeling are captured.
3. **Plugin for annotating process description:** This plugin is used to match standard's requirements with processes. This plugin contains an extended copy of the tasks defined in the previous plugin (by using *contributes* to the original ones) in the method authoring of EPF composer. Then, tasks are annotated with compliance effects. In the process authoring of EPF composer, the delivery process is modeled and annotated with the compliance effects, which are extracted from the rules defined in the rule set. Once the process is modelled and annotated, an activity diagram (by using the proprietary activity diagram provided by EPF composer) is also created.

The three plugins are exported, to be able to transform their information to the inputs required by Regorous.

### 10.2.2 Modelling and annotating a small example from ISO 26262

In this section, the mechanism to model and annotate software processes using EPF Composer is presented by modeling a simple example from ISO 26262, Part 6. Initially, a plugin for capturing standard's requirements is created. For this, a custom category root called Standard Requirements ISO 26262 Software Unit Design is defined, to which the requirements from ISO 26262 are adhered. The standard's requirements, which are depicted in Figure 104 and Figure 106 are represented with a short but descriptive name in a nested list of custom categories. Then the rules (a subset of the rules modeled in Section 10.1.4 and presented in Figure 108) are associated to the corresponding requirements.

```


$$r_1 := [OM]addressSwUnitDesignProcess$$


$$r_2 : addressSwUnitDesignProcess$$


$$\Rightarrow [OANPNP] - performSpecifySwUnit$$


$$r'_2 : performProvideSwArchitecturalDesign,$$


$$performProvideSwSafetyRequirements$$


$$\Rightarrow [P]performSpecifySwUnit$$


$$r_3 : performSpecifySwUnit$$


$$\Rightarrow [OANPNP]selectMandatoryNotationsforSwDesign$$


$$r'_3 : provideRationaleForNotSelectMandatoryNotationsforSwDesign$$


$$\Rightarrow [P] - selectMandatoryNotationsforSwDesign$$

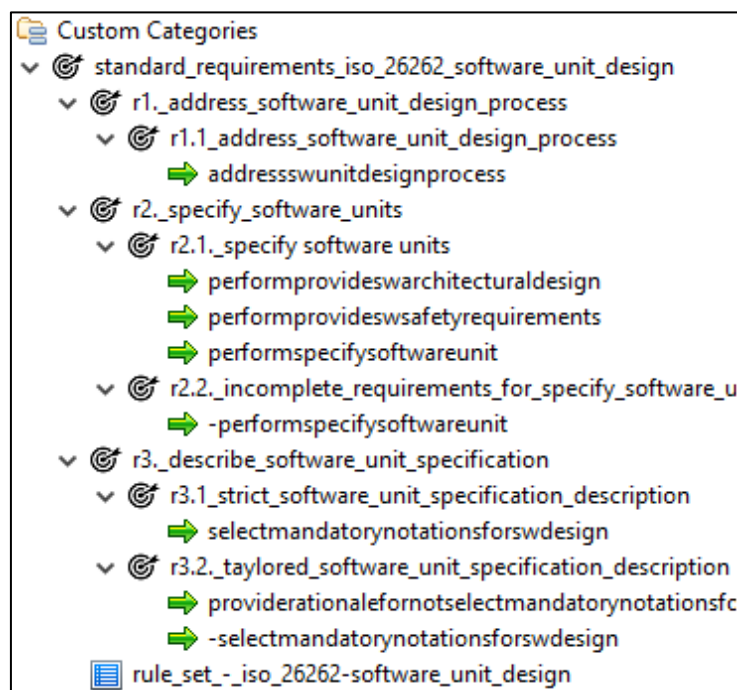

$$r'_2 > r_2$$


$$r'_3 > r_3$$


```

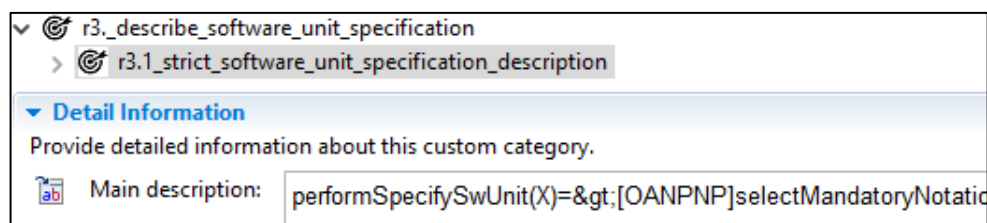
**Figure 108.** Rules required for compliance checking of a small example in ISO 26262

The customized list of standard's requirements and the rules are depicted Figure 109.



**Figure 109.** Standard's requirements plugin

The actual rule is written in the main description field of the compliance effect (see Figure 110).



**Figure 110.** Specification of rule 3.1

The rule set is defined in a customized reusable asset (called Rule Set-ISO 26262-Software Unit Design), which contains the superiority relations between rules (see Figure 111).

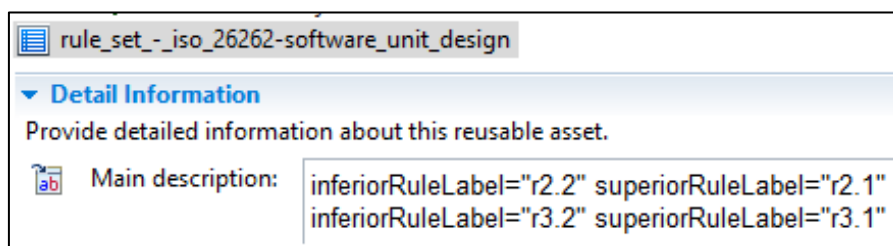


Figure 111. Rule set specification

Then, the plugin for capturing process elements is created (See Figure 112).

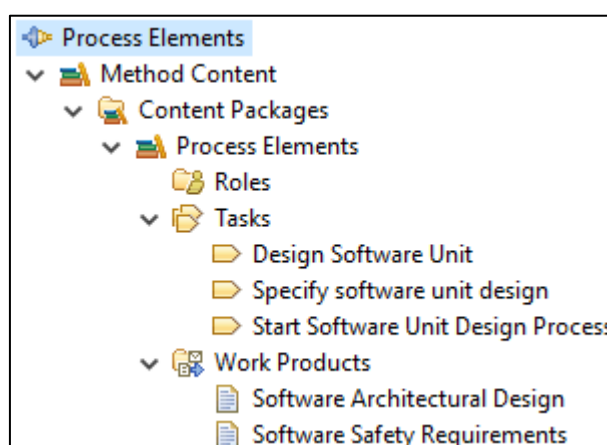


Figure 112. Process elements plugin

Finally, the third plugin is created. In this plugin a copy of the process tasks is carried out. The copied tasks are extended (with contributes) to the original tasks. 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 we initiate addressing the process. This task has two inputs, i.e., *the software safety requirements and the architectural design*. Thus, it also produces the compliance effects *performProvideAssociatedSwSafetyRequirements* and *performProvideSwArchitecturalDesign*. The annotation can be seen in the section called Concepts (See Figure 113).


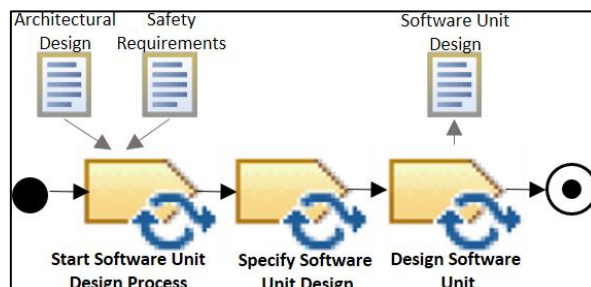
Task: Start Software Unit Design Process	
	
Relationships	
Inputs	Mandatory: <ul style="list-style-type: none"> <li>Software Architectural Design</li> <li>Software Safety Requirements</li> </ul>
Process Usage	<ul style="list-style-type: none"> <li>Software Unit Design Process &gt; Start Software Unit Design Process</li> </ul>
More Information	
Concepts	<ul style="list-style-type: none"> <li>addressSoftwareUnitDesignAndImplementationPhase</li> <li>performProvideAssociatedSoftwareSafetyRequirements</li> <li>performProvideSoftwareArchitecturalDesign</li> </ul>

Figure 113. Annotated task

Then, the annotated tasks (modelled in the method content of the plugin) are used to describe the breakdown structure and the activity diagram. The activity diagram, which represents the delivery process, is depicted in Figure 114.



**Figure 114.** Activity Diagram of the Software Unit Design Process

Once created, the three plugins are exported. From the exported plugins, two files are extracted.

- First, an XMI file (usually called diagram), which describes the activity diagram, is selected for transferring the process description required by Regorous. The elements of interest are an *Activity* that provides the name of the process, an *initial node* and a *final node* that represent the start and end event respectively, one *Activity parameter node* for every task and one *control flow* for every sequence. Other process elements were not modeled in the example presented. However, the file can also provide a *decision*, *merge*, and *fork* and *join nodes* for modeling *exclusive* and *parallel gateway* respectively, which can be useful for complex processes.
- Second, an XML file, which provides the compliance annotated process information, is also extracted. In Table 17, the elements required for compliance checking are presented. As the table shows, the activity name corresponds to the process name. Tasks have associated concepts that correspond to the compliance effects. We can also create the rule set since every concept is described with the actual rule and the reusable asset with the superiority relation.

**Table 17.** Annotated process description

Element	Information
Activity name	Software Unit Design Process
Task use name	Start Software Unit Design Process
- Concept	addressSwUnitDesignProcess performProvideAssociatedSwSafetyRequirements PerformProvideSwarchitecturalDesign
Task use name	Specify Software Unit Design
-Concept	performSpecifySoftwareUnit
Task use name	Design Software Unit
-Concept	SelectMandatoryNotationsForSwDesign

## 10.3 Generating Regorous inputs

In this section, it is defined the transformation (surrounded by a yellow line in Figure 107) necessary to automatically generate the models required by Regorous, i.e., the FCL rule set, the structural representation of the process and the compliance effects annotations [100].

### 10.3.1 Generating the rule set

As recalled in Section 8.1.6, Regorous requires a *rule set* that conforms to the Regorous schema, called *Combined Rule Set*. This information can be obtained from the *Delivery Process* provided by EPF Composer and

described with UMA elements, as recalled in Section 8.1.5. The corresponding mapping descriptions are presented in Table 18.

**Table 18.** Mapping Elements from UMA to the Rule Set

UMA	Rule Set	Mapping Description
Reusable Asset	Rule Set	Reusable Asset, a type of content element, is transformed into the rule set. The attributes transferred are <i>name</i> , <i>presentationName</i> and <i>briefDescription</i> .
Concept	Term	Concept, a type of content element, is transformed into the Terms. The attribute transferred is <i>name</i> .
Content Category	Rule	Each content category that contains a rule in the field brief description is transformed into a rule. The attributes transferred are <i>name</i> , <i>presentationName</i> , and <i>briefDescription</i> .

The algorithmic solution for obtaining the rule set is presented in Figure 115. The algorithm initiates with the description of its required input (DeliveryProcess), which is loaded with the function LoadFunction, and the expected output (RuleSet). Then the input is parsed with the function getElementByTagName, which searches the elements to be mapped, with the function Map to the output. The first element searched is the uma:ReusableAsset, whose attribute name is mapped to the rules uri. Then, the algorithm searches for the elements uma:ContentCategory, which provides the attributes id, controlObjective and formalRepresentation of each rule.

```

input : DeliveryProcess
output: RuleSet
LoadFile (DeliveryProcess);
NodeReusableAsset←getElementByTagName (uma:ReusableAsset);
Map (ruleSet←ReusableAsset);
conceptList←getElementByTagName (uma:Concept);
for i ← 0 to getLength (ConceptList) do
    | Map (Term.atom ← Concept.name)
end
contentCategoryList ← getElementByTagName (uma:ContentCategory);
for j ← 0 to getLength (contentCategoryList) do
    | ruleControlObjective←getAttribute (briefDescription);
    | if ruleControlObjective is not empty then
        | Map (rule←contentCategory)
    end
end

```

**Figure 115.** Algorithm for Obtaining the Rule Set

### 10.3.2 Generating the structural representation of the process

Regorous also requires the representation of the process, which currently is produced by using a subset of BPMN 2.0. Within AMASS, such representation is given via EPF Composer-supported representation, which is based on UML 2.0 Diagram Interchange Specification. In Table 19, the mapping between BPMN 2.0, CPF, and UML is given.

**Table 19.** Mapping Elements from UML Diagram to a BPMN and Canonical Process

BPMN	CPF	UML	Mapping description between UML and CPF
process	Canonical Process	Activity	Activity information is transformed to a canonical process in CPF. The attribute transferred is <i>id</i> .
startEvent	Start Event	Initial Node	The Initial Node becomes a node with type start event in CPF.



BPMN	CPF	UML	Mapping description between UML and CPF
userTask	Task Type	Activity Parameter Node	Each Activity Parameter Node becomes a task type in the CPF. Attributes transferred are id and name.
sequenceFlow	Edge	Control Flow	Each Control Flow becomes an edge in CPF. Attributes transferred are id, name, source and target.
endEvent	End Event	Activity Final Node	The Activity Final Node becomes an end event type in CPF.
exclusive gateway	XOR Split Type	Decision Node	The Decision Node becomes an XORSplitType in CPF.
exclusive gateway	XOR Join Type	Merge Node	The Merge Node becomes an XORJoinType in CPF.
parallel gateway	AND Split Type	Fork Node	The Fork Node becomes an ANDSplitType in CPF.
parallel gateway	AND Join Type	Join Node	The Join Node becomes an ANDJoinType in CPF.

Figure 116 describes the algorithmic solution for mapping the elements described in Table 19. The algorithm initiates with the description of its required input (UML Activity Diagram) and the expected output (Canonical Format). The function LoadFile makes the input available for processing. The function getElementByTagName searches specific elements in the file. Initially we search for the tag that corresponds to the element marked as `uml:Activity`, which is mapped (using the function Map) to a process. Using the same function, the nodes tagged as `uml:node` are searched to be mapped to its corresponding element in the output, namely `uml:ActivityParameterNode` is mapped to the `TaskType`, `uml:InitialNode` is mapped to the `startEvent`, `uml:ActivityFinalNode`, is mapped to the `endEvent`, `uml:ForkNode` and `JoinNode`, are mapped to the `parallelGateway`, and `uml:DecisionNode` and `uml:MergeNode`, are mapped to the `exclusiveGateway`. Finally, the nodes tagged as `uml:edge` are used to describe the `sequenceFlow` information required in the process. The mapping of the process structural elements requires a unique identifier (or Id) that is generated internally each time the function Map is used.



```

input : UMLActivityDiagram
output: CanonicalFormat
LoadFile (UMLActivityDiagram);
NodeActivity←getElementsByTagName (uml:Activity) ;
Map (CanonicalProcess← NodeActivity);
nodesList←getElementsByTagName (uml:node);
for i ← 0 to getLength (nodesList) do
    if nodeType=uml:ActivityParameterNode then
        | Map (TaskType←node)
    end
    if nodeType=uml:InitialNode then
        | Map (StatEvent←node)
    end
    if nodeType=uml:ActivityFinalNode then
        | Map (EndEvent←node)
    end
    if nodeType=uml:ForkNode then
        | Map (ANDSplitType←node)
    end
    if nodeType=uml:JoinNode then
        | Map (ANDJoinType←node)
    end
    if nodeType=uml:DecisionNode then
        | Map (XORSplitType←node)
    end
    if nodeType=uml:MergeNode then
        | Map (XORJoinType←node)
    end
end
edgesList←getElementsByTagName (uml:edge);
for j ← 0 to getLength (edgeList) do
    | Map (Edge←edge)
end

```

**Figure 116.** Algorithm for Obtaining the Process Structure

### 10.3.3 Generating the Compliance Effect Annotations

Finally, the compliance effects annotations require a structure that complies with the Regorous schema called *Compliance Check Annotations*. This information can be retrieved from EPF Composer taking into account that the process elements can be extracted from the process structure (described with UML elements) and the compliance effects annotations can be extracted from the delivery process (described with UMA elements). The corresponding matching elements description is presented in Table 20.

**Table 20.** Mapping Elements from UMA/UML Metamodel to the Compliance Check

UML/UMA	Compliance Annotations	Mapping Description
Reusable Asset	ruleSet	This element comes as an UMA element. A reusable asset becomes a ruleSetList. The attribute transferred is the <i>name</i> .
edge	conditions	This element comes as an UML element. Each edge becomes a special element in the compliance annotations file called conditions. The attribute transferred is the <i>id</i> .
node	Task Effects	This element comes as an UML element. The node becomes a Task Effects. The attribute transferred is the <i>id</i> . Then, the id is also used to search for the concepts that should be converted into the compliance effects in the delivery process file.
Concept	Effect	This element comes as an UMA element. Every concept associated to the task is transferred to the Effect. The attribute transferred is the <i>name</i> .

Figure 117 describes the solution for mapping the elements presented Table 20. The algorithm initiates with the description of the required inputs, i.e., *UML Activity Diagram* and the *DeliveryProcess*, and the expected output, i.e., *ComplianceEffectsAnnotations*. After the files are loaded, the algorithm searches in the delivery process the element tagged as *uma:ReusableAsset* and maps it to the rule set. Similarly, the algorithm searches for the elements tagged as *uml:edge* and *uml:node* in the *UMLActivityDiagram* and maps them to the conditions and taskEffects respectively. The node id is used to search for the elements tagged as *uma:concept* in the *DeliveryProcess*, which is mapped to the effects. The previous algorithms were programmed in Java, obtaining the correct formats required by Regorous.

```

input : UMLActivityDiagram,DeliveryProcess
output: ComplianceEffectsAnnotations
LoadFile (UMLActivityDiagram, DeliveryProcess) ;
NodeReusableAsset (from DeliveryProcess)← getElementsByTagName (uma:ReusableAsset) ;
Map ((ruleSet← ReusableAsset) ;
edgesList(from UMLProcess)← getElementsByTagName (uml:edge);
for i ← 0 to getLength (edgeList) do
    | Map (conditions← edge)
end
nodeList(from UMLProcess)← getElementsByTagName (uml:node);
for j ← 0 to getLength (nodeList) do
    Map (taskEffects← node) TaskId← ObtainUMAValue(nodeList);
    ContentElementList(from DeliveryProcess)←
        getElementsByTagName(ContentElement);
    for k ← 0 to getLength (ContentElementList) do
        if ContentElementList.id = TaskId then
            ConceptsList(from DeliveryProcess)← getElementsByTagName(Concept);
            for l ← 0 to getLength (ConceptsList) do
                | Map (effects← Concept);
            end
        end
    end
end

```

**Figure 117.** Algorithm for Obtaining the Compliance Effects Annotations

### 10.3.4 Model checkable for compliance: an example for ISO 26262

In what follows, the transformations are applied to the example of annotated process, described in Section 10.2.2. The first model corresponds to the *generated Rule Set*. As presented in Figure 118, the generated Rule Set has the elements *Vocabulary*, which contains the rules, described in EPF Composer with an *uma:concept*. It also contains the *rules* described in the *content category elements*, that correspond to the rules.



```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RuleSet xmlns="http://www.nicta.com.au/bpc/CombinedRuleSetDefinition/0.1" uri="rule_set_-_iso_26262-software_unit">
  <Vocabulary>
    <Term atom="performspecifysoftwareunit"/>
    <Term atom="addressswunitdesignprocess"/>
    <Term atom="performprovideswarchitecturaldesign"/>
    <Term atom="performprovideswsafetyrequirements"/>
    <Term atom="selectmandatorynotationsforswdesign"/>
    <Term atom="providerationalefornotselectmandatorynotationsforswdesign"/>
  </Vocabulary>
  <Rules>
    <Rule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="DflRuleType" ruleLabel="r1.1">
      <ControlObjective>r1.1 Address software unit design process</ControlObjective>
      <FormalRepresentation>=>[OANPP]addressSwUnitDesignProcess(X)</FormalRepresentation>
    </Rule>
    <Rule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="DflRuleType" ruleLabel="r2.1">
      <ControlObjective>r2.1. Complete requirements for specifying software units</ControlObjective>
      <FormalRepresentation>performProvideSwArchitecturalDesign(X),performProvideSwSafetyRequirements(X)=>[OANPP]selectMandatoryNotationsforSwDesign(X)</FormalRepresentation>
    </Rule>
    <Rule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="DflRuleType" ruleLabel="r3.1">
      <ControlObjective>r3.1 Strict software unit specification description</ControlObjective>
      <FormalRepresentation>performSpecifySwUnit(X)=>[OANPP]selectMandatoryNotationsforSwDesign(X)</FormalRepresentation>
    </Rule>
    <Rule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="DflRuleType" ruleLabel="r3.2">
      <ControlObjective>r3.2. Tailored software unit specification description</ControlObjective>
      <FormalRepresentation>provideRationaleForNotSelectMandatoryNotationsforSwDesign(X)=>[P]-selectMandatoryNotationsforSwDesign(X)</FormalRepresentation>
    </Rule>
    <Rule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="DflRuleType" ruleLabel="r2.2">
      <ControlObjective>r2.2. Incomplete requirements for specify software units</ControlObjective>
      <FormalRepresentation>addressSwUnitDesignProcess(X)=>[OANPP]-performSpecifySwUnit(X)</FormalRepresentation>
    </Rule>
  <SuperiorityRelations>
    <SuperiorityRelation inferiorRuleLabel="r2.1" superiorRuleLabel="r2.2"/>
    <SuperiorityRelation inferiorRuleLabel="r3.1" superiorRuleLabel="r3.2"/>
  </SuperiorityRelations>
</RuleSet>

```

**Figure 118.** Rule set generated

In Figure 119 it is presented the generated process structure. As the figure depicts, there is one *Node* that represents the start point of the process and three nodes that represents task *types*. Three *edges* represent the connection between the nodes.

In Figure 120, the compliance annotations generated model is presented. In the model it is possible to see the rule set URI (Uniform Resource Identifier), which is the rule set identification, conditions element id, which represent control flows identification, and the taskEffects represent the tasks, whose effects name corresponds to the actual effects.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns4:CanonicalProcess xmlns:ns2="http://www.nicta.com.au/bpc/ComplianceCheckAnnotations/0.3.1" xmlns:ns3="http://
xmlns:ns4="http://www.apromore.org/CPF" name="IdSoftwareUnitDesignProcess">
  <Net id="1529500343213">
    <Node xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns4:EventType" id="1529500343214">
      <name>Start</name>
      <attribute typeRef="bpmnId" value="startevent1"/>
    </Node>
    <Node xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns4:TaskType" id="1529500343215">
      <name>Start Software Design Process</name>
      <attribute typeRef="bpmnId" value="StartSoftwareDesignProcessID"/>
    </Node>
    <Node xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns4:TaskType" id="1529500343216">
      <name>Specify Software Unit Design</name>
      <attribute typeRef="bpmnId" value="SpecifySoftwareUnitDesignID"/>
    </Node>
    <Node xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns4:TaskType" id="1529500343217">
      <name>Design Software Unit</name>
      <attribute typeRef="bpmnId" value="DesignSoftwareUnitID"/>
    </Node>
    <Edge id="1529500343228" default="false" sourceId="1529500343214" targetId="1529500343215">
      <attribute typeRef="bpmnId" value="flow1"/>
    </Edge>
    <Edge id="1529500343229" default="false" sourceId="1529500343215" targetId="1529500343216">
      <attribute typeRef="bpmnId" value="flow2"/>
    </Edge>
    <Edge id="1529500343230" default="false" sourceId="1529500343216" targetId="1529500343217">
      <attribute typeRef="bpmnId" value="flow3"/>
    </Edge>
  </Net>
</ns4:CanonicalProcess>
```

Figure 119. Process structure generated

```
<?xml version="1.0" encoding="ASCII"?>
<cca:ComplianceAnnotations xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:cca="http:
<ruleSetList>
  <ruleSets uri="rule_set_-_iso_26262-software_unit_design"/>
</ruleSetList>
  <conditions elementId="_jNj1AEExVEeiW4M4duzOA6Q"/>
  <conditions elementId="_jukQUExVEeiW4M4duzOA6Q"/>
  <conditions elementId="_J6_jwGANEeit35gRvItDpw"/>
  <conditions elementId="_KVtw4GANEeit35gRvItDpw"/>
  <taskEffects elementId="_hCKUcExVEeiW4M4duzOA6Q">
    <effects name="addressswunitdesignprocess"negation="false">
    <effects name="performprovideswarchitecturaldesign"negation="false">
    <effects name="performprovideswsafetyrequirements"negation="false">
  <taskEffects elementId="_hCKUdExVEeiW4M4duzOA6Q">
    <effects name="performspecifysoftwareunit"negation="false">
  <taskEffects elementId="_hCKUeExVEeiW4M4duzOA6Q">
    <effects name="selectmandatorynotationsforswdesign"negation="true">
    <effects name="providerationalefornotselectmandatorynotationsforswdesign"negation="false">
  <taskEffects elementId="_IsOrEGsEEeiznqPN9kgzVA">
  <localVocabulary/>
</cca:ComplianceAnnotations>
```

Figure 120. Compliance annotations generated



## 11. Implementation solution for compliance checking: a way forward

In this chapter, a way forward concerning the implementation is proposed. The real implementation is expected to be given in D6.6 [21], output of Task 6.3. Table 21 provides details concerning the new functionality for the AMASS Compliance Management Vision.

**Table 21.** Compliance checking

ID	Short Description	Description	Prototype Nº	Priority	Elaborated in section
WP6_CM_004	Triggering compliance checking	The AMASS tools shall provide the functionality for automatically triggering the requirements for (re)checking the compliance of safety processes against rules – especially, when there is change in the standards/ regulations.	P2	shall	Chapter 5
WP6_CM_009	Process Compliance (formal) management	The AMASS tools shall enable users to formally check process compliance.	P2	shall	Chapter 8÷10

## 12. Conceptual solution for ontology-based mapping (\*)

### 12.1 Representation of Safety Standards with Semantic Technologies Used in Industrial Environments (\*)

Understanding and following safety standards with their text can be difficult. Ambiguity and inconsistency, among other issues, can easily arise. As a solution, several authors argue for the explicit representation of the standards with models, which can be created with semantic technologies such as ontologies. However, this possibility has received little attention. The few authors that have addressed it have also only dealt with a subset of safety standard aspects and have used technologies not usually applied for critical systems engineering. As a first step towards addressing these issues, we are working on the representation of safety standards with Knowledge Manager (KM) [140], a tool used in industrial environments that exploits semantic technologies to manage domain information.

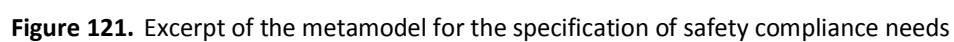
Our proposal to represent safety standards with semantic technologies is based on two main elements: KM, as supporting approach and tool for semantic specification of a standard's information, and a holistic generic metamodel for the specification of safety compliance needs [141]. The metamodel indicates the element types that must be considered when having to demonstrate compliance with safety standards, as well as the relationships between them. The overall purpose of our proposal is to provide guidance about how a standard's terminology, data items of the element types, and relationships between the items can be represented with KM.

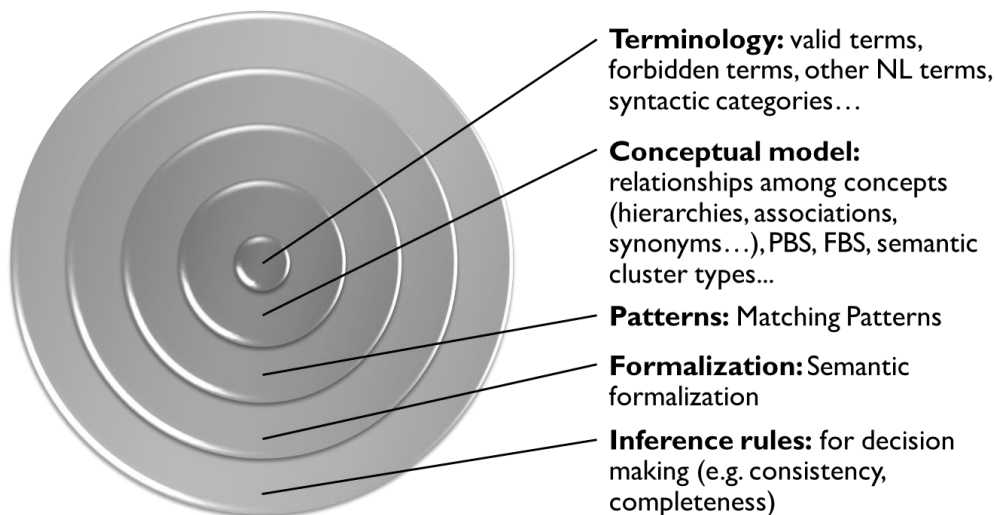
An excerpt of the metamodel is shown in Figure 121. The metamodel supports the specification of the different types of safety compliance needs: information about safety assurance requirements, artefacts, and activities, and about their applicability. This also includes additional information about roles, techniques, artefact attributes, artefact relationships, and relationships between the element types. All the classes in the metamodel specialise *Reference Element*, and *Reference Activity*, *Reference Artefact*, *Reference Role*, and *Reference Technique* specialise *Constrained Reference Assurable Element*. Further information about the metamodel can be found in [141].

Figure 122 shows the structure of an ontology in KM. An ontology consists of several layers, each depending on and extending the semantic information of the inner layer. The most inner 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. At its current state, the proposal only deals with the Terminology and the Conceptual model layers. KM is available in [140].

The proposal consists of two main activities: KM configuration and specification of a standard's information. Each activity consists of several steps, as we explain below. We have already applied the proposal for certain parts of DO-178C, EN 50128, and ISO 26262.







**Figure 122.** Ontology layers in KM

**1. 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. Two tasks must be performed.

**1.1 Specification of semantic clusters.** New clusters must be added to the Conceptual model layer to be able to indicate the type of information that a term represents. First, a cluster with the name of the safety standard to be represented is necessary to later specify that a term falls within the scope of the standard. Second, semantic clusters must be added for Reference Artefact, Reference Artefact Attribute, Reference Activity, Reference Role, and Reference Technique, a cluster for each. These clusters are part of another new cluster called Reference Assurance Framework. The semantic clusters will be used to further categorise certain terms.

**1.2 Specification of relationship types.** KM also supports the specification of relationship types between terms. To represent a safety standard, a relationship type has to be created for each association in the metamodel between the metaclasses for which the new clusters have been added, e.g. for 'user-inputArtefact' between Reference Activity and Reference Artefact. This does not apply to the compositions, e.g. between Reference Artefact and Reference Artefact Attribute. KM has a predefined relationship type for composition, as well as for specialisation (to specify e.g. taxonomies) and for equivalence (to specify e.g. synonyms), among others. Another relationship type called 'Reference Artefact Relationship' must be added to be able to relate different Reference Artefacts in KM. The specification of the relationship types also includes the specification of the roles of the relationship ends.

**2. 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.

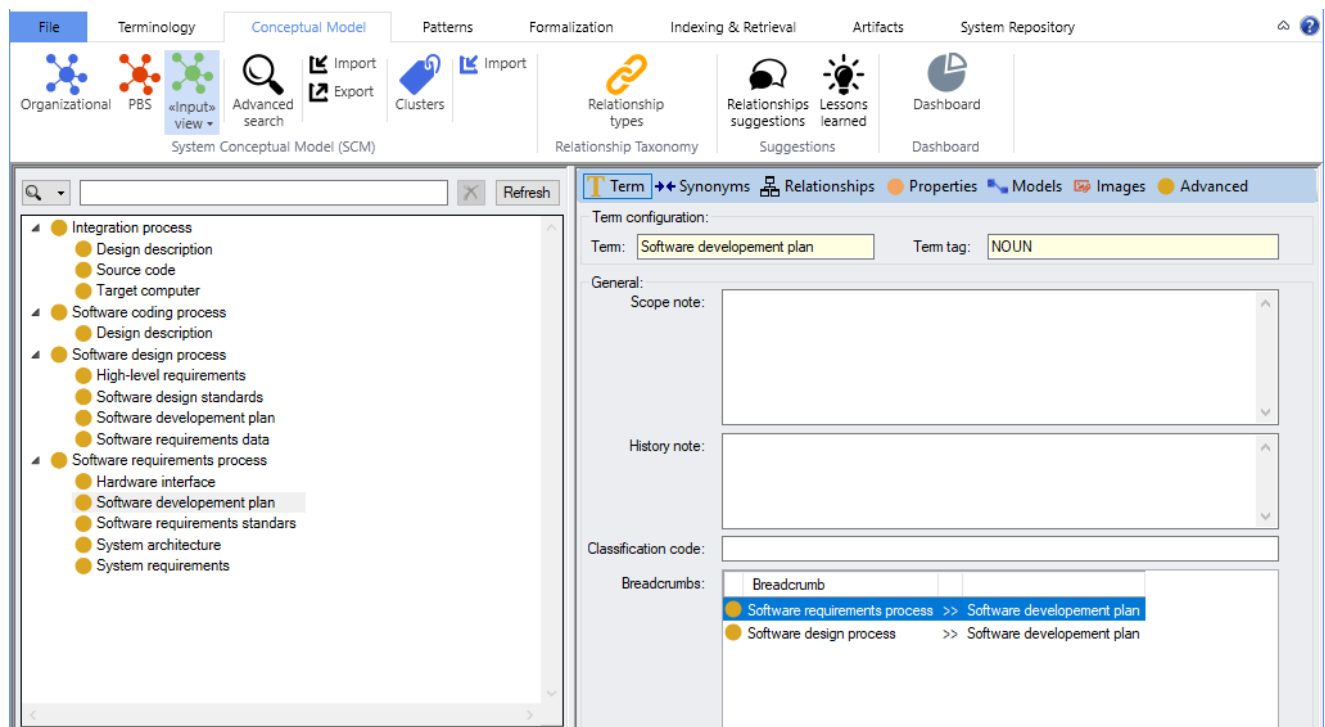
**2.1 Specification of a standard's terminology.** This task has two main aspects to address. First, most standards have some glossary or vocabulary section. The corresponding terms and definitions, abbreviations, and acronyms must be added to the Terminology. Each time a term is added, it is necessary to (1) specify its syntactic category (e.g. noun or acronym) and (2) associate it with the semantic cluster that corresponds to the name of the standard; e.g. the term 'algorithm' would be added as a DO-178C noun. Next, the text of the standard must be analysed to identify terms that correspond to Reference Artefact, Reference Artefact Attribute, Reference Activity, Reference Role, or Reference Technique. Each time a term is identified, it is added to the Terminology and, in addition to

the clusters for the glossary terms, the semantic cluster of the element type is associated; e.g. 'Software Requirements Data' is a DO-178C noun that also corresponds to a Reference Artefact.

**2.2. Specification of the conceptual model of a safety standard.** Once all the relevant terms have been introduced and classified, relationships between them can be specified in the Conceptual model. These relationships will be classified according to the available relationship types in KM, both the default ones and those created during KM configuration. A user must conform to the holistic generic metamodel when specifying relationships, i.e. only terms that correspond to the ends of a given association in the metamodel must be related. For example, 'Software Requirements Data' is an 'output' of 'Software Requirements Process' in DO-178C.

The user also needs to decide whether the relationships between Reference Artefacts should be specified as specialisations, as compositions, or with the Reference Artefact Relationship type. It is also possible to define specialisations of this relationship type if a user decides so, e.g. because it is a recurrent Reference Artefact Relationship. For instance, it is common that artefacts have to 'conform to' some plan or standard. Finally, it can also be necessary to specify specialisation and equivalence relationships between terms; e.g. 'MC/DC' and 'Modified Condition/ Decision Coverage' are equivalent for DO-178C.

Figure 123 shows a part of the representation for DO-178C that results from the application of the approach.



**Figure 123.** Example of specification of a standard's information with Knowledge Manager

Within the overall purpose of demonstrating alignment or compliance with a safety standard, we currently envisage six main possibilities to take advantage of the representations:

- 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.
- 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.

Methodological guidance to enact these scenarios will be provided in D6.8. It is also possible that, as a result of the definition of this guidance and of its application in the industrial use cases, we discover that further benefits can be exploited.

## 12.2 Semantic Analysis of Safety Standards (\*)

Semantic analysis of safety standards (or in general of assurance standards) is an important area for AMASS because it can facilitate tasks such as the interpretation of standards and their comparison. These tasks can later represent a basis for assurance reuse, as the gained insights can guide an engineer when deciding whether it is possible or advisable that a certain assurance asset is reused across products or domains.

No new, specific means for semantic analysis of safety standards are currently envisioned, but AMASS will exploit the support provided by other solutions already defined:

- Equivalence mapping, presented in Section 5.2, which allows an engineer to establish the degree of correspondence between the elements of different standards.
- Semantic representation of safety standards with KM and the usage possibilities that it enables, as described in Section 12.1.

These two solutions can further be applied together when KM is integrated with the AMASS Tool Platform (joint work of WP5 and WP6), and more concretely when CACM models are indexed and stored with KM technology. This requires that KM connects to CDO, which is the main storage technology of the AMASS Tool Platform. Once KM can search in CACM models, which include equivalence maps, and using also the semantic representation of the standards as support, it will be possible to synergistically combine the semantic analysis possibilities of both solutions. In addition, the semantic representation could also assist engineers when specifying equivalence maps, as the representation of different standards can be the basis to decide upon the degree of correspondence between elements.



## 13. Implementation solution for the ontology-based compliance management vision: a way forward

In this chapter, a way forward concerning the implementation is proposed. The real implementation is expected to be given in D6.6 [21], output of Task 6.3. Table 22 provides details concerning the ontology-based compliance management functionality, on top of the semantics-based mapping of standards, for the AMASS Compliance Management Vision.

**Table 22.** Ontology-based compliance management

ID	Short Description	Description	Prototype Nº	Priority	Elaborated in section
WP6_SEM_001	Semantics-based mapping of standards	The AMASS tools shall enable the mapping of standards based on their semantics.	P2	shall	Chapter 12

## 14. Metrics for reuse

So far, metrics to measure the effectiveness of family-oriented engineering approaches have been partly neglected. A lack of metrics can impede their adoption. For instance, organizations considering the adoption of Safety-oriented Process Lines (SoPLs) are faced with the upfront questions regarding the selection of the right processes for conversion, to derive the maximum benefits in the shortest time frame.

Resources can be allocated to an endeavour only in the presence of objective justification of the economic benefits. To provide such justification, an appropriate measurement methodology is needed. In this chapter, the GQM+ Strategies model [83] (shortened GQMPS) is used. GQMPS is an extension of the GQM (Goal Question Metric) paradigm [84], a goal-based software implementation and measurement paradigm.

More specifically, in this chapter, in line with and beyond D1.3 [2], GQMPS paradigm is used in order to further elaborate the STO4-Cross/Intra Domain Reuse. The focus is limited to process-related reuse. However, similar reasoning can be transposed to product as well as assurance case-related reuse.

### 14.1 GQMPS for process-related reuse

For sake of clarity it should be pointed out that a revised and extended version of this section was accepted for publication at EuroSPI-2018 [154].

#### 14.1.1 GQMPS

The GQMPS model links measurement programs to higher level organization goals and strategies [83]. GQMPS is built as an extension of the GQM paradigm, a top down approach, in which measurements are based on measurement goals [84].

As recalled in D1.3 [2], the GQM paradigm consists of three levels: the *conceptual level* (Measurement Goal) where the objectives are defined, the *operational level* (Question) where the questions are defined and the *quantitative level* where the metrics are defined. These levels are also hierarchically organized in a pyramid structure. The apex of the pyramid is represented by a measurement goal, which specifies the purpose of measurement, the object which is being measured, the issue to be measured and the viewpoint from which the measurement is taken. This measurement goal is refined by a set of questions which breaks down the goal into its significant elements. Each question is further refined into one or more metrics. These metrics may either be objective or subjective in nature. Moreover, a particular metric may be used to answer more than one question.

The GQMPS model helps organizations to align multi-level organization goals and strategies to the measurement goals. It consists of two perspectives, the Organizational and Planning Perspective (OPP) and the Control Perspective (CP). The OPP and CP structures help incorporating dependencies among different levels of the organization. The OPP is concerned with the organizational goals and strategies while the CP is concerned with the measurements.

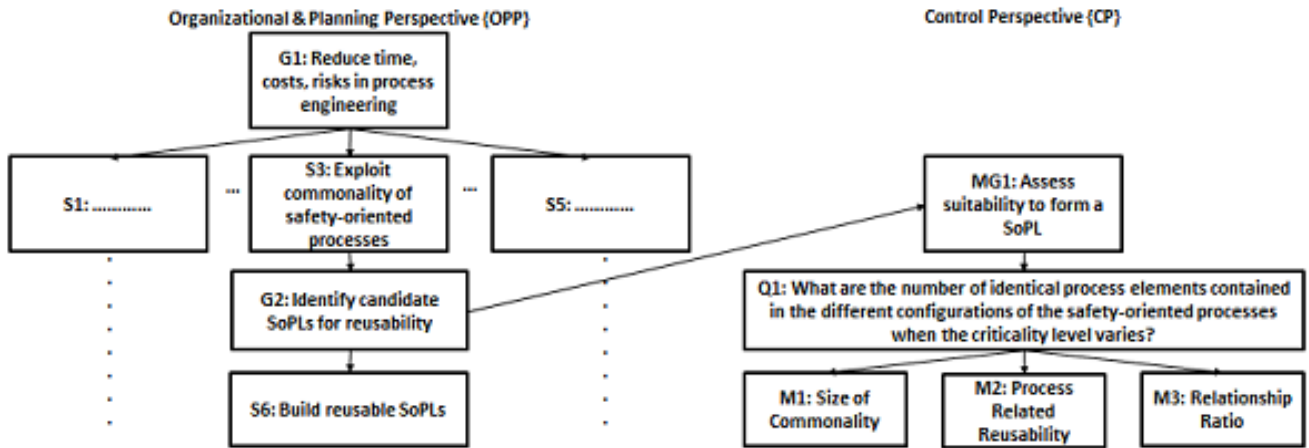
The structure of the OPP resembles a pyramid with the top goal of the organization at the apex. The top goal is broken down into one or more strategies. Each strategy can be further split into one or more goals and associated strategies until the strategies cannot be further split into lower goals. The CP structures are built using the GQM paradigm. Each organizational goal is linked to a GQM structure in the CP via a measurement goal. These links enable alignment of organizational goals and strategies with measurement goals. This ensures that organizations invest resources only in meaningful and essential data collection and analysis activities.

#### 14.1.2 GQM + Strategies Model for the evaluation of families of processes (\*)

In this section, a GQMPS model for the evaluation of families of safety-oriented processes (SoPLs) is developed. This model represents an initial design of the measurement framework.



The overview of this model is given in Figure 124.



**Figure 124.** SoPLE-targeted GQM+ Strategies Model

The model includes only the goals related to the feasibility of establishing SoPLs.

The OPP structure reflects the organizational goals and strategies starting with the top overall organization goal and the related strategies broken down below, reflecting the SoPLE organization goals and their related strategies. The association of the top organizational goal (G1) to the strategy (S3) of 'Exploit commonality of safety-oriented processes' is given. The strategy S3 is further reduced to the software development organization (SDO) goal G2, that of 'identify candidate SoPLs for reusability' supporting the strategy S6, 'Build reusable SoPLs'. Strategy S3 is reduced to a single SDO goal for illustration purposes, though it may be reduced to additional goals, such as productivity and quality-related goals.

The CP part of the model links the goal G2 to the measurement goal MG1, 'Assess suitability to form a SoPL'. The viewpoint is that of the SoPL manager who belongs to the SDO and has overall organizational responsibility for software development with formation of SoPLs as the object. MG1 is progressively refined to question Q1 and the metrics M1, M2 and M3 addressing the extent of commonality.

The model builds on top of metrics which were developed within the product line community. More specifically, in [85], Berger et al. define several metrics which provide different perspectives for assessing the suitability for setting up product lines. Among such metrics, within the proposed model (Figure 61), three metrics are used: Size of Commonality (SoC), Process-related Reusability (PrR), and Relationship Ratio (RR).

Originally, SoC measures the number of reusable components in a product line and is determined by comparing the component signatures. A syntactic comparison of signatures is performed from the names of the components, while a semantic comparison is performed from the behavioural profiles of the components that capture behavioural constraints. If the two signatures are identical, the components are identical. SoC is computed as shown in Equation (1a) where  $p_i$  represents the products of the product line,  $i$  ranges from 2 to  $n$  and  $C_{pi}$  represents the set of components of the product  $i$ .

$$SoC = \left| \bigcap_{i=1}^n C_{pi} \right| \quad (1a)$$

PrR measures the extent of reusability of the common components for a specific product. PrR is computed as shown in Equation (1b).

$$PrR_i = \frac{SoC}{|C_{pi}|} \quad (1b)$$

RR measures the extent of commonality between pairs of products of the product line. RR is computed as shown in Equation (1c) where  $i$  and  $j$  range from 1 to  $n$ ,  $n$  is greater than 1, and  $C_{pi}$  and  $C_{pj}$  represent the set of components of the  $i$  and  $j$  products respectively.



$$RR_{i,j} = \frac{|C_{pi} \cap C_{pj}|}{|C_{pi} \cup C_{pj}|} \quad (1c)$$

The model developed in this chapter has been applied for measuring the gain obtained through the AMASS solutions in the context of UC11. The application was documented in a paper (see [154]), accepted at EuroSPI conference. The further development is planned in the context of T6.4, D6.8 [23].

## 14.2 GQMPS for product-&-assurance case related reuse (\*)

Similar to what proposed for measuring process-related reuse, GQMPS customizations can be elaborated for product as well as assurance case-related reuse. Such customizations would exploit the same metrics interpreted in the contexts of either products or assurance cases.



## 15. Conclusion

This deliverable has first recalled the context/motivation and AMASS-specific needs concerning cross-and-intra domain reuse and compliance management.

Then, it has documented the final design of the AMASS solutions for cross-and-intra domain reuse (in Chapters 3-6), offering also a set of metrics for evaluating the potential gain implied by the adoption of the proposed solutions (Chapter 14).

This deliverable has also documented the final design of the AMASS solutions for compliance management (Chapter 7-13).

It should be highlighted that both designs comprise a conceptual and tool-agnostic solution, to be deployed within the AMASS platform. It should also be highlighted that the proposed solutions have been illustrated via a rich set of simplified examples stemming from the AMASS case studies and that, in the majority of the cases, these solutions have been presented in relevant venues and accepted for publication.

In the remaining period of the AMASS project, the focus will be on: 1) the finalization of the implementation of the presented design solutions (to be made available as part of the AMASS final prototype and documented in D6.6 [21]); 2) the definition of the methodological guidelines (to be documented in D6.8 [23]), where the potential user (less interested in the design choices) gets guidance on how and which methods/tools best fit together in the context of an application. Finally, cooperation with WP1 for the demonstration of the effectiveness of the proposed solutions is also expected to take place in the remaining period.



## Abbreviations and Definitions

ACM	Association for Computing Machinery
ACS	Attitude Control System
ADAS	Advanced Driver-Assistance System
AOCS	Attitude Orbit Control Systems
API	Application Programming Interface
ARTA	AMASS Reference Tool Architecture
ASIL	Automotive Safety Integrity Level
BPMN	Business Process Model and Notation
BVR	Base Variability Resolution
BCL	Basic Constraint Language
CACC	Cooperative Adaptive Cruise Control
CACM	Common Assurance and Certification Meta-model
CAKE	Computer-Aided Knowledge Environment
CAN	Controller Area Network
CBSE	Component-Based Software Engineering
CCL	Common Certification Language
CCU	Central Computing Unit
CDO	Connected Data Objects
CDT	Contrary-to-Duty Obligations
CHESSML	CHESS Modelling Language
COTS	Commercial Off The Shelf
CP	Control Perspective
CPF	Canonical Process Format
CPS	Cyber Physical System
CVL	Common Variability Language
DSL	Domain Specific Language
DUI	Delegated User Interface
EASA	European Aviation Safety Agency
ECSS	European Cooperation for Space Standardization
ECU	Electronic Control Unit
ECMP	Electronic Component Management Process
EMC	Electromagnetic compatibility
EMF	Eclipse Modelling Framework
EPF	Eclipse Process Framework
EPS	Electrically Assisted Power Steering
ERMTS	European Rail Traffic Management System
ESD	Electrostatic Discharge
ETL	Epsilon Transformation Language
FAA	Federal Aviation Administration
FCL	Formal Contract Logic
FDIS	Final Draft International Standard
FLEDS	Fuel Level Estimation and Display System
FMEDA	Failure Modes Effects (Diagnostic) Analysis
FPGA	Field Programmable Gate Array
FODA	Feature-Oriented Domain Analysis
FCL	Formal Contract Logic
FTA	Fault Tree Analysis
GQM	Goal Question Metric
GQMPS	GQM+ Strategies model



---

GSN	Goal Structuring Notation
HARA	Hazard Analysis Risk Assessment
HTTP	Hypertext Transfer Protocol
HW	Hardware
ICS	Industrial Control Systems
ICU	Integrated Control Unit
IMA	Integrated Modular Avionics
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
LIN	Local Interconnect Network
KM	Knowledge Manager
KPI	Key Performance Indicator
MBSE	Model-based Systems Engineering
MCU	Microcontroller Unit
MDE	Model Driven Engineering
MOF	Meta Object Facility
MOTS	modified off the shelf
NATO	North Atlantic Treaty Organization
NICTA	National ICT Australia
OCL	Object Constraint Language
OCRA	Othello Contracts Refinement Analysis
OEM	Original Equipment Manufacturer
OMG	Object Management Group
OPP	Organizational and Planning Perspective
OSLC	Open Services for Lifecycle Collaboration
OTS	Off The Shelf
PHA	Preliminary Hazard Analysis
PoS	Part-of-Speech
PrR	Process-related Reusability
RC	Resistor Capacitor
RecL	Recommendation Level
RDF	Resource Description Framework
REST	Representational State Transfer
RPC	Remote Procedure Call
RQS	Requirements Quality Suite
RR	Relationship Ratio
SA	Safety Architect
SACM	Structured Assurance Case Meta-model
SAE	Society of Automotive Engineers
SAS	Systems Assets Store
SDO	Software Development Organization
SecL	Security Levels
SEI	Software Engineering Institute
SEooC	Safety Element out-of-Context
SIL	Safety Integrity Level
SKB	System Knowledge Base
SiSoPL	Security-informed Safety-oriented Process Line
SKR	System Knowledge Repository
SOAP	Simple Object Access Protocol
SoC	Size of Commonality
SoPL	Safety-oriented Process Lines
SoPLE	Safety-oriented Process Lines Engineering



SOUP	Software of Unknown Pedigree
SPeM	Software and Systems Process Engineering Meta-model
SPL	Split Phase Level
SPLCA	Software Product Line Covering Array
SRL	Security Risk Level
SSRW	Sun Sensors Reaction Wheels
SSTH	Sun Sensors Thrusters
STO	Scientific and Technical Objective
STRW	Star Tracker Reaction Wheels
STTH	Star Tracker Thrusters
SUT	System Under Test
SW	Software
SysML	Systems Modelling Language
TARA	Threat Assessment & Risk Analysis
TVM	Transmission Voie-Machine
UDP	User-defined Process
UMA	Unified Method Architecture
UML	Unified Modelling Language
URI	Uniform Resource Identifier
V&V	Verification & Validation
WSDL	Web Services Description Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language





## References

- [1] [AMASS D1.1 Case studies description and business impact](#), 9th February 2018.
- [2] [AMASS D1.3 Evaluation framework and quality metrics](#), 30th September 2017.
- [3] [AMASS D2.1 Business cases and high-level requirements](#), 28th February 2017.
- [4] AMASS D2.2 AMASS reference architecture (a), 30th November 2016.
- [5] AMASS D2.3 AMASS reference architecture (b), 30th September 2017.
- [6] [AMASS D2.4 AMASS reference architecture \(c\)](#), 4th June 2018.
- [7] [AMASS D2.6 Integrated AMASS platform \(a\)](#), 31st March 2017.
- [8] [AMASS D3.1 Baseline and requirements for architecture-driven assurance](#), 9th March 2018.
- [9] AMASS D3.2 Design of the AMASS tools and methods for architecture-driven assurance (a), 30th June 2017.
- [10] [AMASS D3.3 Design of the AMASS tools and methods for architecture-driven assurance \(b\)](#), 31st March 2018.
- [11] [AMASS D3.5 Prototype for architecture-driven assurance \(b\)](#), 30th September 2017.
- [12] AMASS D4.2 Design of the AMASS tools and methods for multi-concern assurance (a), 31st March 2017.
- [13] [AMASS D4.3 Design of the AMASS tools and methods for multi-concern assurance \(b\)](#), 30th April 2018.
- [14] [AMASS D5.1 Baseline requirements for seamless interoperability](#), 30th November 2016.
- [15] AMASS D5.2 Design of the AMASS tools and methods for seamless interoperability (a), 31st March 2017.
- [16] [AMASS D5.3 Design of the AMASS tools and methods for seamless interoperability \(b\)](#), 30th June 2018.
- [17] [AMASS D6.1 Baseline and requirements for cross/intra-domain reuse](#), 9th March 2018.
- [18] AMASS D6.2 Design of the AMASS tools and methods for cross/intra-domain reuse (a), 31st October 2017.
- [19] [AMASS D6.4 Prototype for cross/intra-domain reuse \(a\)](#), 31st March 2017.
- [20] [AMASS D6.5 Prototype for cross/intra-domain reuse \(b\)](#), 31st December 2017.
- [21] AMASS D6.6 Prototype for cross/intra-domain reuse (c), 31st October 2018.
- [22] AMASS D6.7 Methodological guide for cross/intra-domain reuse (a), 31st December 2017.
- [23] AMASS D6.8 Methodological guide for cross/intra-domain reuse (b), 31st October 2018.
- [24] M. F. Johansen, O. Haugen, F. Fleurey, A. G. Eldegard, and T. Syversen. Generating better partial covering arrays by modelling weights on sub-product lines. Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MODELS), Innsbruck, Austria, LNCS, vol 7590. Springer, pp. 269-284, September 30-October 5, 2012.
- [25] I. Ayala, B. Gallina. Towards Tool-based Security-informed Safety Oriented Process Line Engineering. 1st ACM International workshop on Interplay of Security, Safety and System/Software Architecture (ISSA), Copenhagen, Denmark, November 28th, 2016.
- [26] B. Gallina, E. Gómez-Martínez, C. Benac Earle. Promoting MBA in the Rail Sector by Deriving Process-related Evidence via MDSafeCer. Computer Standards & Interfaces -SPICE-2016 Special Issue (CSI SPICE-2016), <http://dx.doi.org/10.1016/j.csi.2016.11.007>.
- [27] J. P. Castellanos Ardila and B. Gallina. Towards Increased Efficiency and Confidence in Process Compliance. 24th European & Asian Systems, Software & Service Process Improvement & Innovation, Ostrava, Czech Republic, 5.-8. Sept. 2017.
- [28] C. Cărlan, B. Gallina, S. Kacianka, R. Breu. Arguing on Software-level Verification Techniques Appropriateness. Proceedings of the 36th International Conference on Computer Safety, Reliability and Security (SAFECOMP), Trento, Italy, September 12-15, 2017.
- [29] B. Gallina. A Model-driven Safety Certification Method for Process Compliance. 2nd IEEE International Workshop on Assurance Cases for Software-intensive Systems (ASSURE), joint event of ISSRE, Naples, Italy, doi: 10.1109/ISSREW.2014.30, pp. 204-209, November 3-6, 2014.



- [30] B. Gallina, K. Lundqvist and K. Forsberg. THRUST: A Method for Speeding Up the Creation of Process-related Deliverables. IEEE 33rd Digital Avionics Systems Conference (DASC-33), doi:10.1109/DASC.2014.6979489, Colorado Springs, CO, USA, October 5-9, 2014.
- [31] O. M. G., "Software & systems process engineering meta-model specification," Object Management Group, Tech. Rep. formal/2008-04-01, April 2008. [Online]. Available: <http://www.omg.org/spec/SPEM/2.0/PDF>
- [32] Ø. Haugen, "Common Variability Language (CVL)," Object Management Group, Tech. Rep. ad/2012-08-05, August 2012. [Online]. Available: <http://www.omgwiki.org/variability/doku.php>
- [33] VARIES," <http://www.varies.eu/>, accessed: 2017-07-13.
- [34] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.
- [35] SACM: <http://www.omg.org/spec/SACM/2.0/Beta1>
- [36] Eclipse process framework project (epf)," <https://eclipse.org/epf/> , accessed: 2017-09-12.
- [37] European cooperation for space standardization, ecss-e-st-40c, space engineering software. [http://www.wis.win.tue.nl/2R690/doc/ECSS-E-ST-40C\(6March2009\).pdf](http://www.wis.win.tue.nl/2R690/doc/ECSS-E-ST-40C(6March2009).pdf). Last accessed: June 16, 2017.
- [38] Ø. Haugen and O. Øgård. BVR - better variability results. Proceedings of the 8th International Conference on System Analysis and Modeling: Models and Reusability (SAM), Valencia, Spain. In D. Amyot, P. Fonseca i Casas, and G. Mussbacher, editors, System Analysis and Modeling: Models and Reusability, volume 8769 of Lecture Notes in Computer Science, pages 1–15, Springer International Publishing, 2014.
- [39] M. Jones, E. Gomez, A. Mantineo, and U. K. Mortensen. Introducing ECSS software-engineering standards within ESA. [http://www.esa.int/esapub/bulletin/bullet111/chapter21\\_bul111.pdf](http://www.esa.int/esapub/bulletin/bullet111/chapter21_bul111.pdf), August 2002.
- [40] A. Vasilevskiy and Ø. Haugen. Resolution of interfering product fragments in software product line engineering. In J. Dingel, W. Schulte, I. Ramos, S. Abrahão, and E. Insfran, editors, Model-Driven Engineering Languages and Systems, volume 8767 of Lecture Notes in Computer Science, pages 467–483. Springer International Publishing, 2014.
- [41] Øystein Haugen. Variability modelling. <https://wiki.hiof.no/images/3/34/CPS-11-VariabilityModeling.pdf>
- [42] VARIES D4.2- BVR - The language. [http://bvr.modelbased.net/docs/VARIES\\_D4.2\\_v01\\_PP\\_FINAL.pdf](http://bvr.modelbased.net/docs/VARIES_D4.2_v01_PP_FINAL.pdf)
- [43] VARIES D4.3 BVR Tool. [http://www.varies.eu/wp-content/uploads/sites/8/2013/05/VARIES\\_D4.3\\_v01\\_PU\\_FINAL.pdf](http://www.varies.eu/wp-content/uploads/sites/8/2013/05/VARIES_D4.3_v01_PU_FINAL.pdf)
- [44] EPF Composer Architecture Overview. [https://www.eclipse.org/epf/composer\\_architecture/](https://www.eclipse.org/epf/composer_architecture/)
- [45] ISO/IEC 15026-2:2011, Systems and software engineering -- Systems and software assurance -- Part 2: Assurance case
- [46] Governatori, G., & Sadiq, S. (2008). The Journey to Business Process Compliance. Public Law, 1-32.
- [47] Antoniou, G., Billington, D., Governatori, G., Maher, M. (2000). Representation Results for Defeasible Logic. ACM Transactions on Computational Logic, 255-287.
- [48] Governatori, G., Rotolo, A., Sartor, G. (2005). Temporalised Normative Positions in Defeasible Logic. 10th International Conference on Artificial Intelligence and Law (ICAIL), 25-34.
- [49] CVL, OMG Revised Submission, <http://www.omgwiki.org/variability/lib/exe/fetch.php?media=cvl-revised-submission.pdf>
- [50] OCL, <http://www.omg.org/spec/OCL/2.4/>
- [51] I. Sljivo, B. Gallina, J. Carlson, H. Hansson, S. Puri. A Method to Generate Reusable Safety Case Fragments from Compositional Safety Analysis. Proceedings of the 14th International Conference on Software Reuse (ICSR), Springer, LNCS 8919, ISBN 978-3-319-14130-5, pp.253-268, Miami, Florida, USA, January 4-6, 2015.



- [52] B. Gallina, L. Fabre. (2015, September). Benefits of security-informed safety-oriented process line engineering. In *Digital Avionics Systems Conference (DASC)*, 2015 IEEE/AIAA 34th (pp. 8C1-1). IEEE.
- [53] B. Gallina, S. Kashiyyarandi, 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.
- [54] D. Nute, "Defeasible Logic," in *International Conference on Applications of Prolog*, 2001, pp. 151–169.
- [55] V. W. Marek and M. Truszczyński, *Nonmonotonic logic: context-dependent reasoning*. Springer Science & Business Media, 2013.
- [56] R. Reiter, "A logic for default reasoning," *Artif. Intell.*, vol. 13, no. 1–2, pp. 81–132, 1980.
- [57] W. Lukaszewicz, "Considerations on Default Logic: An Alternative Approach," *Comput. Intell.*, vol. 4, no. 1, pp. 1–16, 1988.
- [58] G. Brewka, "Cumulative default logic: In defense of nonmonotonic inference rules," *Artif. Intell.*, vol. 50, no. 2, pp. 183–205, 1991.
- [59] G. Gottlob and M. Zhang, "Cumulative Default Logic: Finite Characterization, Algorithms, and Complexity," *Artif. Intell.*, vol. 69, no. 1–2, pp. 329–345, 1994.
- [60] J. P. Delgrande, T. Schaub, and W. K. Jackson, "Alternative approaches to default logic," *Artif. Intell.*, vol. 70, no. 1–2, pp. 167–237, 1994.
- [61] A. Mikitiuk and M. Truszczyński, "Constrained and rational default logics," in *International Joint Conference on Artificial Intelligence*, 1995, pp. 1509–1517.
- [62] R. C. Moore, "Semantical considerations on nonmonotonic logic," *Artif. Intell.*, vol. 25, no. 1, pp. 75–94, 1985.
- [63] J. McCarthy, "Circumscription - a form of Non-monotonic Reasoning," *Artif. Intell.*, vol. 13, no. 1, pp. 27–39, 1980.
- [64] M. Maher, "Propositional Defeasible Logic has Linear Complexity," *Theory Pract. Log. Program.*, vol. 1, no. 6, pp. 691–711, 2001.
- [65] G. Antoniou, D. Billington, G. Governatori, and M. J. Maher, "Representation results for defeasible logic," *ACM Trans. Comput. Log.*, vol. 2, no. 2, pp. 255–287, 2001.
- [66] D. Nute, *Defeasible Deontic Logics*. Springer Science & Business Media, 2012.
- [67] G. H. Von Wright, "Deontic logic," *Mind*, vol. 60, no. 237, pp. 1–15, 1951.
- [68] G. Governatori, "Representing business contracts in RuleML," *Int. J. Coop. Inf. Syst.*, vol. 14, no. 02n03, pp. 181–216, 2005.
- [69] G. Governatori and A. Rotolo, "A Gentzen System for Reasoning with Contrary-To-Duty Obligations. A Preliminary Study," in *Sixth International Workshop on Deontic Logic in Computer Science*, 2002, pp. 97–116.
- [70] G. Governatori and A. Rotolo, "A conceptually rich model of business process compliance," in *7th Asia-Pacific Conference on Conceptual Modelling*, 2010, vol. 110, pp. 3–12.
- [71] G. Governatori and Z. Milosevic, "Dealing with contract violations: Formalism and domain specific language," *Proc. - IEEE Int. Enterp. Distrib. Object Comput. Work. EDOC*, pp. 46–57, 2005.
- [72] 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.
- [73] G. Governatori, F. Olivieri, A. Rotolo, and S. Scannapieco, "Computing Strong and Weak Permissions in Defeasible Logic," *J. Philos. Log.*, vol. 42, no. 6, pp. 799–829, 2013.
- [74] Nicta, "Regorous Process Designer Help."
- [75] G. Governatori and S. Sadiq, "The Journey to Business Process Compliance," *Public Law*, pp. 1–32, 2009.
- [76] 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.
- [77] Object Management Group, "Documents Associated With Business Process Model And Notation™ (BPMN™) Version 2.0." Available: <http://www.omg.org/spec/BPMN/2.0/>. [Accessed 2017/09/13].



- [78] Polygraphia, "Ontotext," [Online].  
Available: <https://ontotext.com/knowledgehub/fundamentals/semantic-annotation/>.  
[Accessed 2017/09/11].
- [79] S. Sadiq, G. Governatori, and K. Namiri, "Modeling Control Objectives for Business Process Compliance," *5th Int. Conf. BPM*, pp. 149–164, 2007.
- [80] H. Lam and G. Governatori, "The Making of SPINdle," *Rule Interchang. Appl.*, pp. 315–322, 2009.
- [81] R. Lu, S. Sadiq, and G. Governatori, "Compliance Aware Business Process Design," in *International Conference on Business Process Management*, 2007, pp. 120–131.
- [82] B. Gallina, J. P. Castellanos Ardila, and M. Nyberg, "Towards Shaping ISO 26262-compliant Resources for OSLC-based Safety Case Creation," in *4th International Workshop on Critical Automotive Applications: Robustness & Safety*, 2016, p. 4.
- [83] V. Basili, A. Trendowicz, M. Kowalczyk, J. Heidrich, C. Seaman, J. Munch, and D. Rombach. GQM+ Strategies in a Nutshell. In *Aligning Organizations Through Measurement*, pages 9–17. Springer, 2014.
- [84] V. R. Basili, G. Caldiera, and H. D. Rombach. The Goal Question Metric Approach, 1994. Cited, 3:11, 2012.
- [85] C. Berger, H. Rendel, and B. Rumpe. Measuring the Ability to Form a Product Line from Existing Products. arXiv preprint arXiv:1409.6583, 2014.
- [86] 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.
- [87] B. Gallina and M. Nyberg, "Reconciling the ISO 26262-compliant and the agile documentation management in the Swedish context," in *CARS 2015 - Critical Automotive applications: Robustness & Safety*, Paris, 2015.
- [88] J. P. Castellanos Ardila and B. Gallina, "Towards Increased Efficiency and Confidence in Process Compliance," in *24th European & Asian Systems, Software & Service Process Improvement & Innovation*, 2017, p. 12.
- [89] J. P. Castellanos-Ardila and B. Gallina, "Towards Efficiently Checking Compliance Against Automotive Security and Safety Standards," in *The 7th IEEE International Workshop on Software Certification*, 2017.
- [90] ISO 26262, "Road Vehicles-Functional Safety. International Standard." 2011.
- [91] A. Koudri and J. Champeau, "MODAL: A SPEM extension to improve co-design process models," *Int. Conf. Softw. Process*, pp. 248–259, 2010.
- [92] B. Gallina, I. Sljivo, and O. Jaradat, "Towards a Safety-oriented Process Line for Enabling Reuse in Safety Critical Systems Development and Certification," in *35th Annual IEEE Software Engineering Workshop (SEW)*, 2012, pp. 148–157.
- [93] D. Riehle and H. Züllighoven, "Understanding and using patterns in software development," *Tapos*, vol. 2, no. 1, pp. 3–13, 1996.
- [94] J. 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*, 2017, pp. 65–72.
- [95] M. Dwyer, G. Avrunin, and J. Corbett, "Property Specification for Finite-State Verification," in *International Conference on Software Engineering*. 1998, pp. 411–420.
- [96] B. McIsaac, "IBM Rational Method Composer: Standards Mapping." 2015.
- [97] J. P. Castellanos Ardila, B. Gallina, and F. U. Muram, "Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models," in *Euromicro Conference on Software Engineering and Advanced Applications*, 2018.
- [98] O. M. Group, "UMA metamodel."  
[http://www.eclipse.org/epf/tool\\_component/EPF\\_Schema\\_201003161045.xsd](http://www.eclipse.org/epf/tool_component/EPF_Schema_201003161045.xsd)
- [99] M. La Rosa et al., "APROMORE: An advanced process model repository," *Expert Syst. Appl.*, pp. 7029–7040, 2011.





- [100] J. P. Castellanos Ardila, B. Gallina, and F. UL Muram, "Transforming SPEM 2.0-compatible Process Models into Models Checkable for Compliance," in *18th International SPICE Conference*, 2018.
- [101] IEC 62443:2015 Security for industrial automation and control systems.
- [102] The European Organisation for Civil Aviation Equipment, "Airworthiness Security Process Specification", ED-202A, 2014.
- [103] The European Organisation for Civil Aviation Equipment, "Airworthiness Security Methods and Considerations", ED-203, 2015.
- [104] The International Electrotechnical Commission, "Industrial-process measurement, control and automation- Framework for functional safety and security", IEC TR 63069 ED1, TC 65, 2018-06.
- [105] The International Electrotechnical Commission, "Functional safety of electrical/electronic/programmable electronic safety-related systems", IEC 61508, 1998.
- [106] McQueen M.A., Boyer W.F., Flynn M.A., Beitel G.A., "Time-to-Compromise Model for Cyber Risk Reduction Estimation." In: Gollmann D., Massacci F., Yautsiukhin A. (eds) *Quality of Protection. Advances in Information Security*, vol 23. Springer, Boston, MA, 2006.
- [107] L. Piètre-Cambacédès and M. Bouissou, "Modeling safety and security interdependencies with BDMP (Boolean logic Driven Markov Processes)," 2010 IEEE International Conference on Systems, Man and Cybernetics, Istanbul, pp. 2852-2861, 2010.
- [108] The Eclipse Foundation, "The Papyrus Environment", in <https://www.eclipse.org/papyrus/download.html>.
- [109] The International Electrotechnical Commission, "Industrial communication networks – Network and system security – Part 1-1: Terminology, concepts and models", IEC/TS 62443-1-1, 2009.
- [110] Automotive Electronics Council, AEC Q100 "Failure Mechanism Based Stress Test Qualification For Integrated Circuits"; <http://www.aecouncil.com/AECDocuments.html>
- [111] International Automotive Task Force, IATF 16949:2016 – Technical Specification "Quality management system requirements for automotive production and relevant service parts organisations"; <http://www.iatfglobaloversight.org/iatf-publications/>
- [112] International Organization for Standardization, ISO 9001:2015 "Quality management systems -- Requirements"; <https://www.iso.org/standard/62085.html>
- [113] [https://www.bav.admin.ch/dam/bav/fr/dokumente/nntv/eisenbahn/nntv\\_loc\\_pas\\_gueltig.pdf.download.pdf/RTNN%20LOC&PAS%20-%20Versione%20actuelle%20\(septembre%202017\).pdf](https://www.bav.admin.ch/dam/bav/fr/dokumente/nntv/eisenbahn/nntv_loc_pas_gueltig.pdf.download.pdf/RTNN%20LOC&PAS%20-%20Versione%20actuelle%20(septembre%202017).pdf)
- [114] <http://www.securite-ferroviaire.fr/sites/default/files/users/reglementations/pdf/sams706v2.pdf>
- [115] B. Gallina, Z. Szatmari. *Ontology-based Identification of Commonalities and Variabilities among Safety Processes*. Proceedings of the 16th International Conference on Product-Focused Software Process Improvement (PROFES), Springer, LNCS 9459, pp. 182-189, ISBN 978-3-319-26843-9, Bolzano, Italy, December 2-4, 2015.
- [116] 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: October 11, 2017).
- [117] 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: October 11, 2017).
- [118] 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>
- [119] H. Mili, F. Mili, and A. Mili, "Reusing software: Issues and research directions," *Softw. Eng. IEEE Trans. On*, vol. 21, no. 6, pp. 528–562, 1995.
- [120] C. W. Krueger, "Software reuse," *ACM Comput. Surv. CSUR*, vol. 24, no. 2, pp. 131–183, 1992.
- [121] M. Smolárová and P. Návrát, "Software reuse: Principles, patterns, prospects," *CIT J. Comput. Inf. Technol.*, vol. 5, no. 1, pp. 33–49, 1997.
- [122] B. W. Boehm, *Software Engineering Economics*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.



- [123] Y. Kim and E. A. Stohr, "Software Reuse: Survey and Research Directions," *J Manage Inf Syst*, vol. 14, no. 4, pp. 113–147, Mar. 1998.
- [124] H. Mili, *Reuse based software engineering: techniques, organization and measurement*. New York: Wiley, 2002.
- [125] T. J. Biggerstaff and C. Richter, "Software Reusability: Vol. 1, Concepts and Models," T. J. Biggerstaff and A. J. Perlis, Eds. New York, NY, USA: ACM, 1989, pp. 1–17.
- [126] J. Fortune and R. Valerdi, "Considerations for successful reuse in systems engineering," in *Space 2008 Conference*, 2008.
- [127] S. F. Model-Driven, K. A. Weiss, E. C. Ong, and N. G. Leveson, "Reusable specification components for model-driven development," in *In Proceedings of the International Conference on System Engineering, INCOSE*, 2003.
- [128] I. Jacobson, M. Griss, and P. Jonsson, *Software Reuse: Architecture, Process and Organization for Business Success*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997.
- [129] E.-A. Karlsson, Ed., *Software Reuse: A Holistic Approach*. New York, NY, USA: John Wiley & Sons, Inc., 1995.
- [130] D. McIlroy, "Mass-produced Software Components," in *Proceedings of Software Engineering Concepts and Techniques*, Garmisch, Germany, 1969, pp. 138–155.
- [131] INCOSE, "Systems Engineering Vision 2020," INCOSE, Technical INCOSE-TP-2004-004-02, 2004.
- [132] U. Shani and H. Broodney, "Reuse in model-based systems engineering," 2015, pp. 77–83.
- [133] W. B. Smith, "3.3.2 Re-Use Libraries Leveraging Model-Based Systems Engineering to greatly increase engineering productivity," *INCOSE Int. Symp.*, vol. 24, no. 1, pp. 298–312, Jul. 2014.
- [134] C. Dumitrescu, R. Mazo, C. Salinesi, and A. Dauron, "Bridging the gap between product lines and systems engineering: an experience in variability management for automotive model based systems engineering," in *Proceedings of the 17th International Software Product Line Conference*, 2013, pp. 254–263.
- [135] O. Gotel et al., "The Grand Challenge of Traceability (v1.0)," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. London: Springer London, 2012, pp. 343–409.
- [136] J. M. Alvarez-Rodríguez, Elena Gallego and J. Llorens, "Reuse of Physical System Models by means of Semantic Knowledge Representation: A Case Study applied to Modelica," in *Proceedings of the 11th International Modelica Conference 2015*, 2015, vol. 1.
- [137] R. Mendieta, J. L. de la Vara, J. Llorens, and J. M. Alvarez-Rodríguez, "Towards Effective SysML Model Reuse," in *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD*, 2017, pp. 536–541.
- [138] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowl Acquis*, vol. 5, no. 2, pp. 199–220, Jun. 1993.
- [139] N. Guarino, "Formal Ontology, Conceptual Analysis and Knowledge Representation," *Int J Hum-Comput Stud*, vol. 43, no. 5–6, pp. 625–640, Dec. 1995.
- [140] The REUSE Company: <https://www.reusecompany.com/>
- [141] 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)
- [142] B. Gallina, Z. Haider, A. Carlsson. *Towards Generating ECSS-compliant Fault Tree Analysis' Results via Concerto FLA*. 2nd International Conference on Reliability Engineering (ICRE), Milan, Italy, December 20–22, 2017.
- [143] <https://www.polarsys.org/chess/publis/CHESSMLprofile.pdf>
- [144] Schieferdecker, I. (2012). *Model-Based Testing*. *Software*, IEEE, 29(1), 14–18.
- [145] Dalal, S., et.al (1999). *Model-based testing in practice*. *Proceedings - International Conference on Software Engineering*, 285–29
- [146] Utting, Mark (2006), *Practical Model-based Testing* ISBN 9780123725011
- [147] [AMASS D1.4 AMASS Demonstrators \(a\)](#), April 2017





- [148] W. S. Greenwell, J. C. Knight, C. M. Holloway, and J. J. Pease, "A Taxonomy of Fallacies in System Safety Arguments," in In Proceedings of the 2006 International System Safety Conference, 2006.
- [149] 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. (in press).
- [150] M.Soden: Dynamische Modellanalyse von Metamodellen mit Operationaler Semantik. Dissertation, 2014
- [151] ANSYS medini Technology AG: medini analyze Metamodel Guide
- [152] B. Kaiser et al.: Advances in Component Fault Trees. To appear in Proceedings of ESREL 2018
- [153] M. A. Javed and B. Gallina. Safety-oriented Process Line Engineering via Seamless Integration between EPF Composer and BVR Tool. In 22nd International Systems and Software Product Line Conference (SPLC), Sept 10-14, Gothenburg, Sweden, in press. ACM Digital Library, 2018.
- [154] B. Gallina and S. Iyer. Towards Quantitative Evaluation of Reuse within Safety-oriented Process Lines. 25th European & Asian Systems, Software & Service Process Improvement & Innovation (EuroSPI), Communications in Computer and Information Science, Springer, pp. 162-174, Bilbao, Spain, 5.-7. Sept. 2018.
- [155] B. Gallina, F. UI Muram, and J. P. Castellanos Ardila. Compliance of Agilized (Software) Development Processes with Safety. Proceedings of the 4th international workshop on agile development of safety-critical software (ASCS), co-located with XP 2018, May 21<sup>st</sup>, Porto, Portugal, 2018.
- [156] I. Sljivo, B. Gallina, J. Carlson, H. Hansson, S. Puri. Tool-Supported Safety-Relevant Component Reuse: From Specification to Argumentation. 23rd International Conference on Reliable Software Technologies (Ada-Europe), Lisbon, Portugal, June 18-22, 2018.
- [157] R. Dardar, B. Gallina, A. Johnsen, K. Lundqvist, and M. Nyberg: Industrial Experiences of Building a Safety Case in Compliance with ISO 26262. In: Proc. Of the 2nd WoSoCER, joint event of the 23rd International Symposium on Software Reliability (ISSRE), Dallas, Texas, USA (Nov. 2012) 349–354
- [158] B. Gallina, A. Gallucci, K. Lundqvist and M. Nyberg. VROOM & cC: a Method to Build Safety Cases for ISO 26262-compliant Product Lines. 32nd International Conference on Computer Safety, Reliability and Security (SAFECOMP) 2013 - Workshop {SASSUR} (Next Generation of System Assurance Approaches for Safety-Critical Systems), Toulouse, France, 2013.
- [159] S. Mazzini, J. M. Favaro, S. Puri, and L. Baracchi, "CHESS: an open source methodology and toolset for the development of critical systems," in Joint Proceedings of the 12th Educators Symposium (EduSymp 2016) and 3rd International Workshop on Open Source Software for Model Driven Engineering (OSS4MDE 2016) co-located with the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), Saint Malo, France, October 3, 2016., 2016, pp. 59–66.
- [160] 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, vol. 96, pp. 105–121, 2014.
- [161] E. Lee, *A Denotational Semantics for Dataflow with Firing*. Electronics Research Laboratory, College of Engineering, University of California, 1997.
- [162] AMASS-site: <https://www.amass-ecsel.eu>.
- [163] J. Chelini et al. Avionics Certification: Back to Fundamentals with Overarching Properties. 9th European Congress Embedded Real Time Software and Systems (ERTS)-2018.



## Appendix A

This appendix provides the non-complete BCL grammar, taken from [41]. The complete definition of the BCL grammar and its semantics can be found in [49] (Section 8.3 Definition of Basic Constraint Language).

For the purpose of this document, this appendix is sufficient to let the reader follow the grammatical specification of the BCL constraints, which were included within the VSpec models.

$\langle \text{BCLExpression} \rangle ::= \langle \text{Existence} \rangle \mid \langle \text{NumRelation} \rangle$

$\langle \text{Existence} \rangle ::= \langle \text{UniExistence} \rangle \mid \langle \text{BinaryExistence} \rangle$

$\langle \text{UniExistence} \rangle ::= \langle \text{VSpec} \rangle \mid \text{not } \langle \text{VSpec} \rangle \mid (\langle \text{Existence} \rangle) \langle \text{VSpec} \rangle$  refers to a Choice or a VClassifier

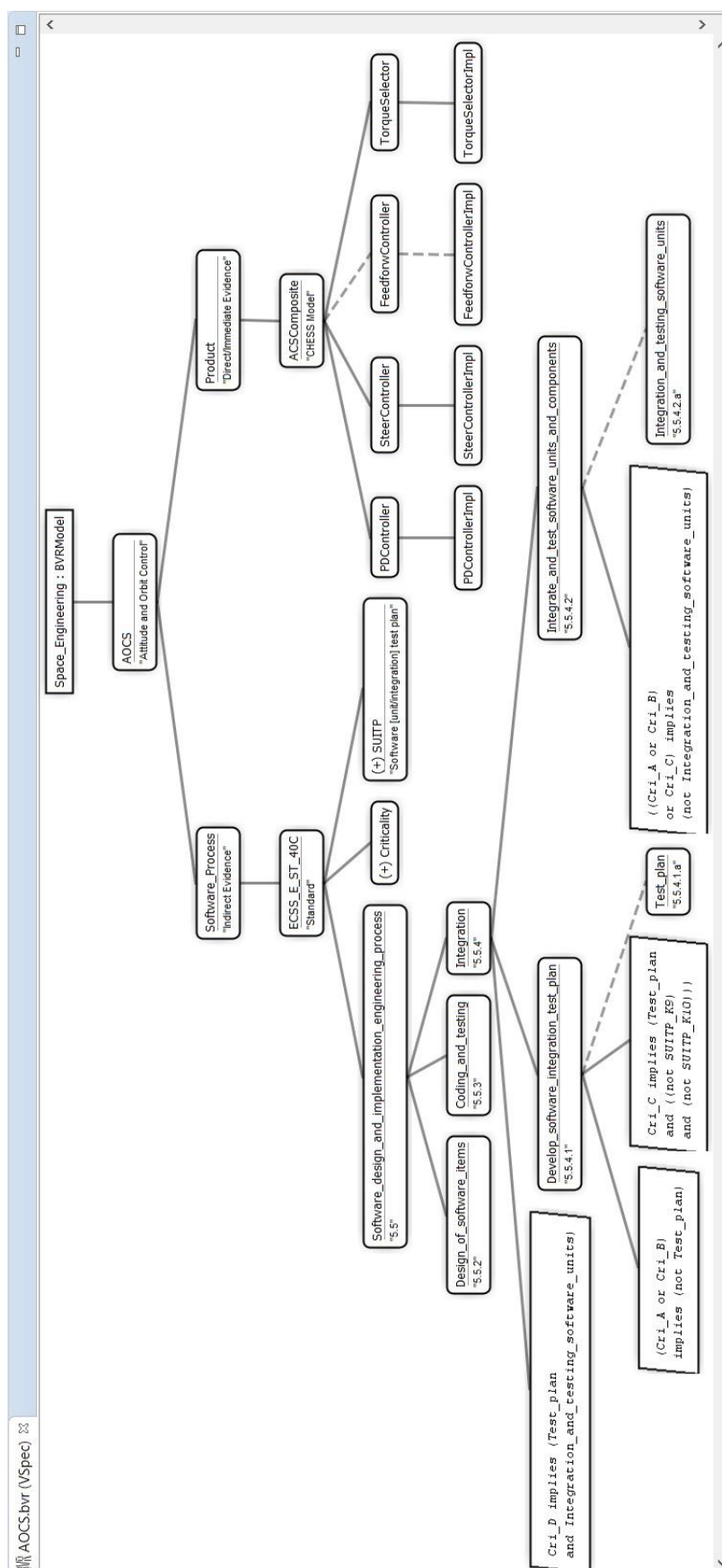
$\langle \text{BinaryExistence} \rangle ::= \langle \text{Existence} \rangle \langle \text{binop} \rangle \langle \text{Existence} \rangle$   $\langle \text{binop} \rangle ::= \text{and} \mid \text{or} \mid \text{xor} \mid \text{implies} \mid \text{iff}$

$\langle \text{NumRelation} \rangle ::= \text{normal arithmetic expression}$



## Appendix B

To enable readers to better read the details of a set of figures, previously presented in the document, this appendix contains their expanded versions.



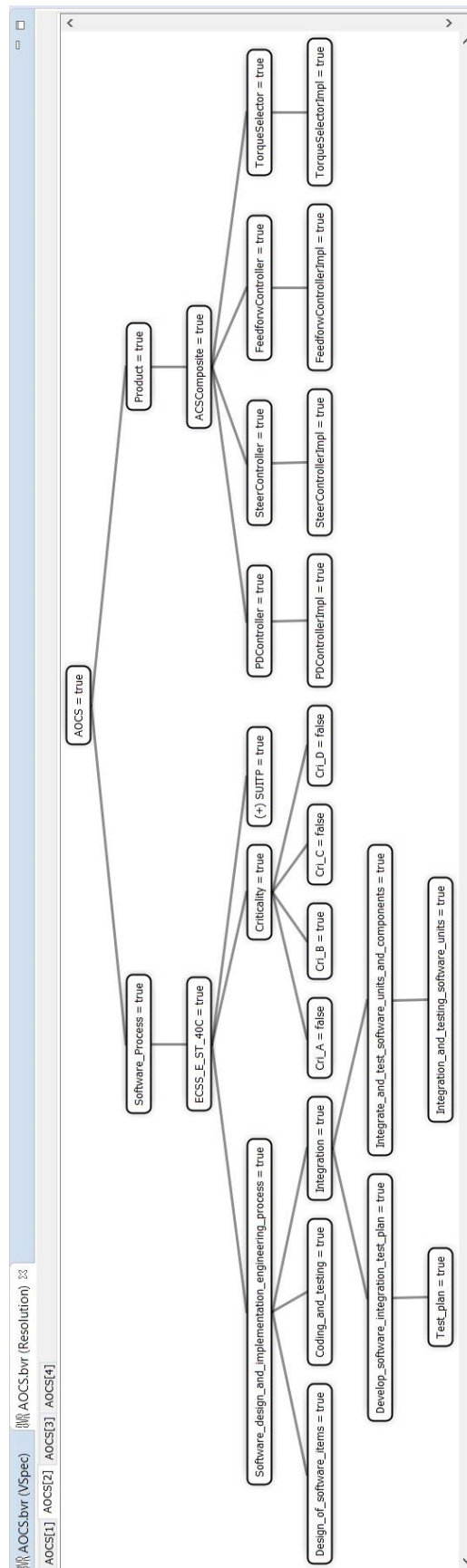


Figure 126. Expanded version of Figure 41

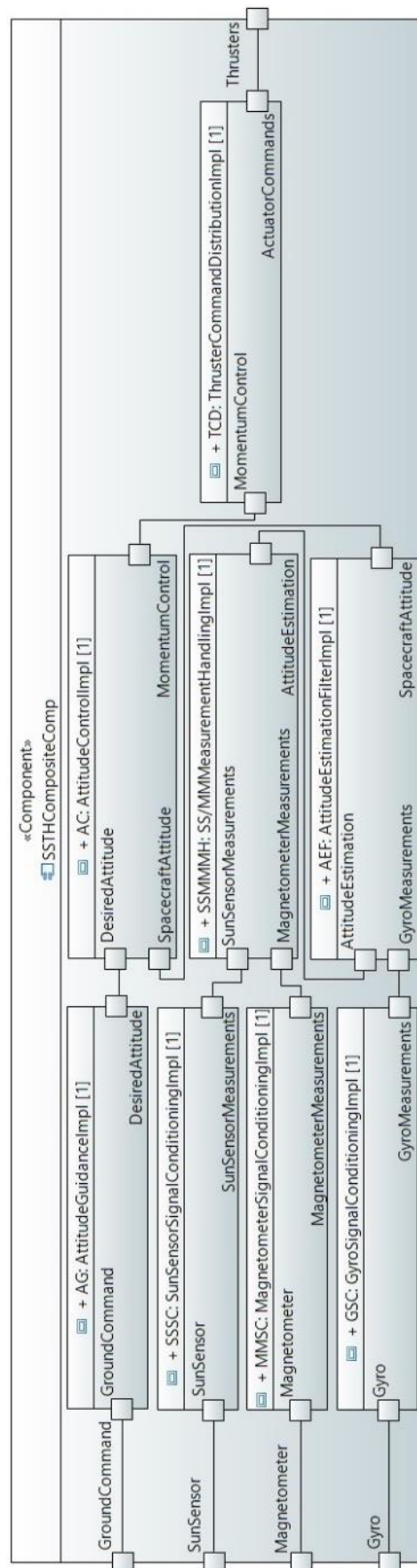


Figure 127. Expanded version of Figure 45



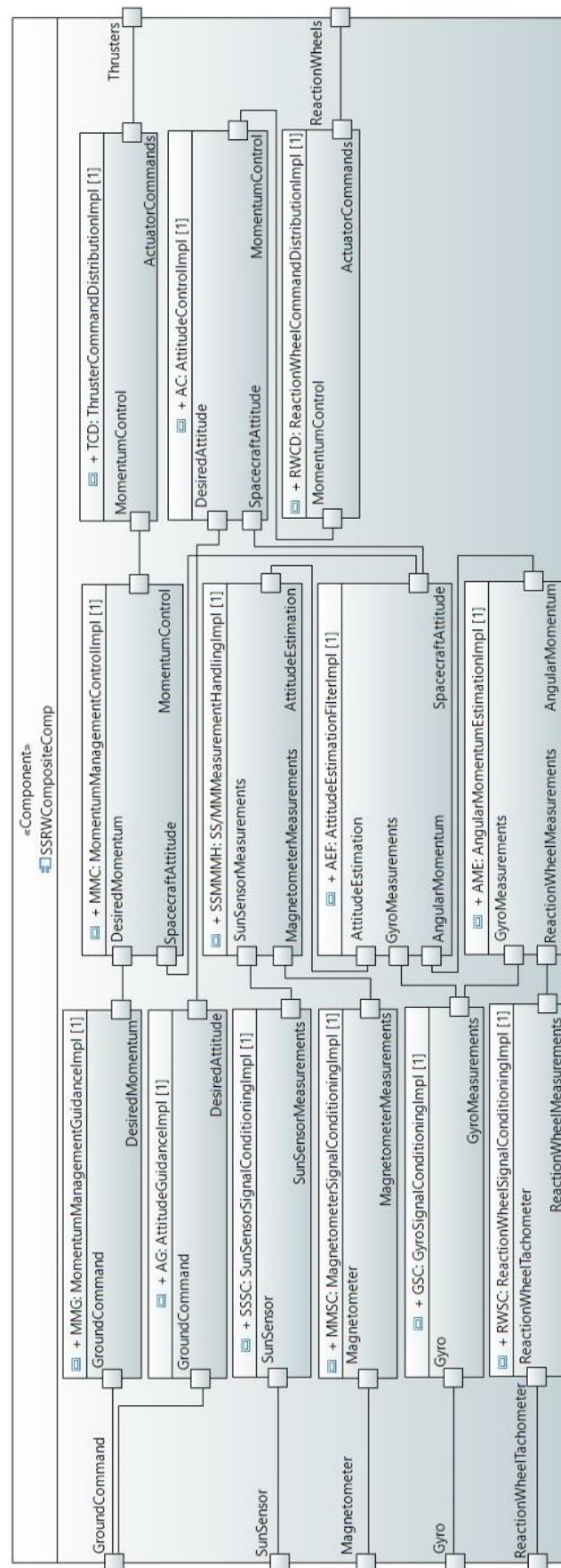


Figure 128. Expanded version of Figure 46

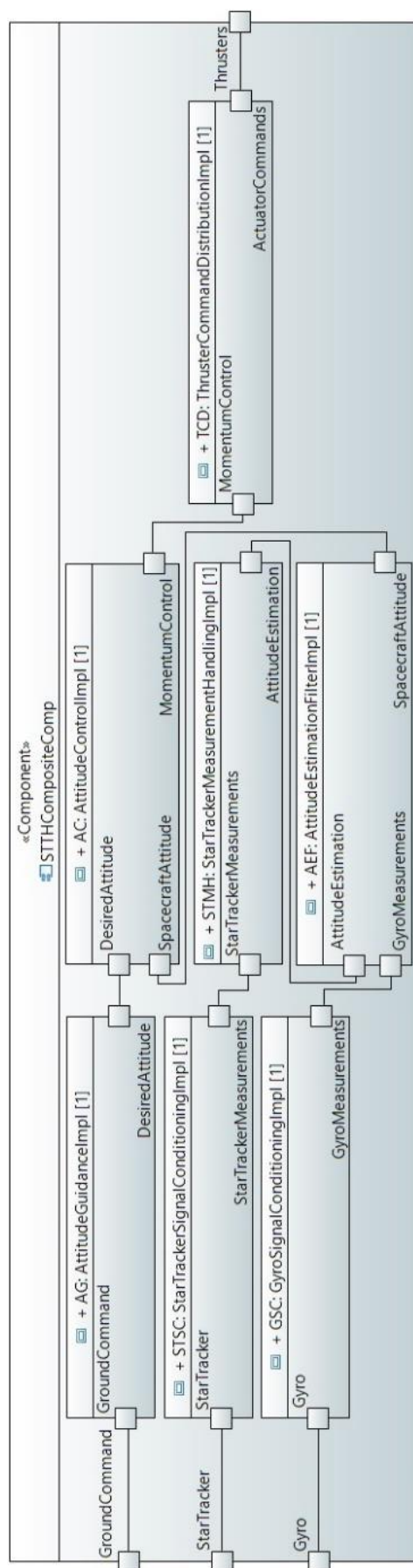


Figure 129. Expanded version of Figure 47

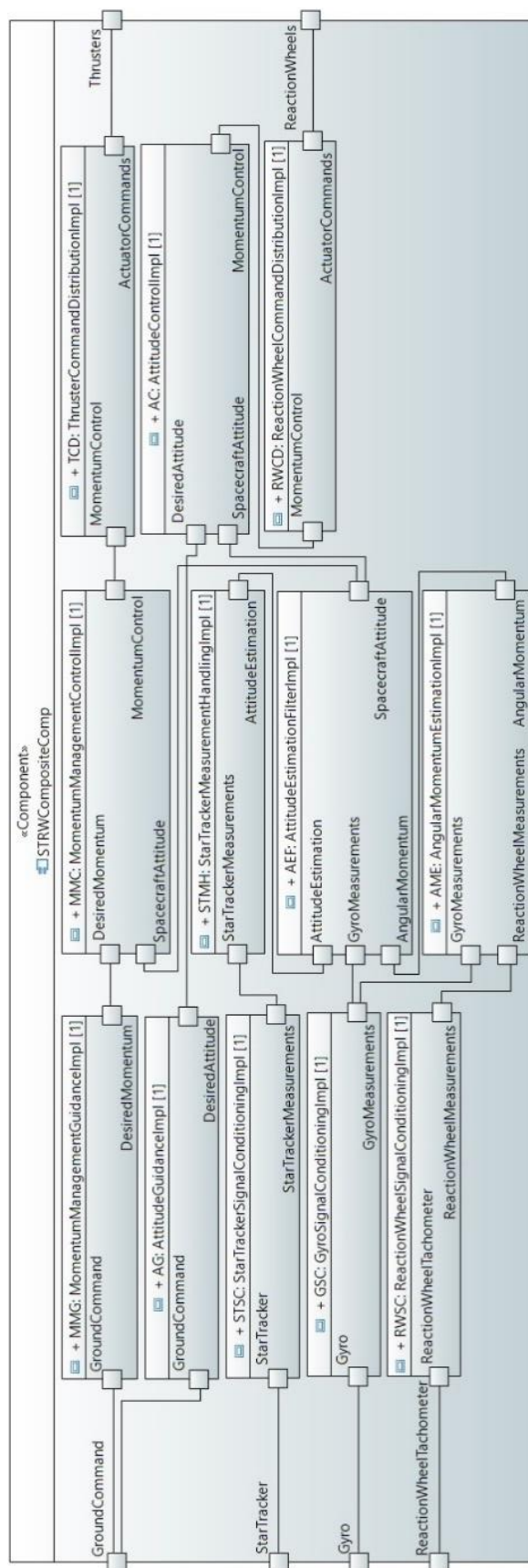


Figure 130. Expanded version of Figure 48

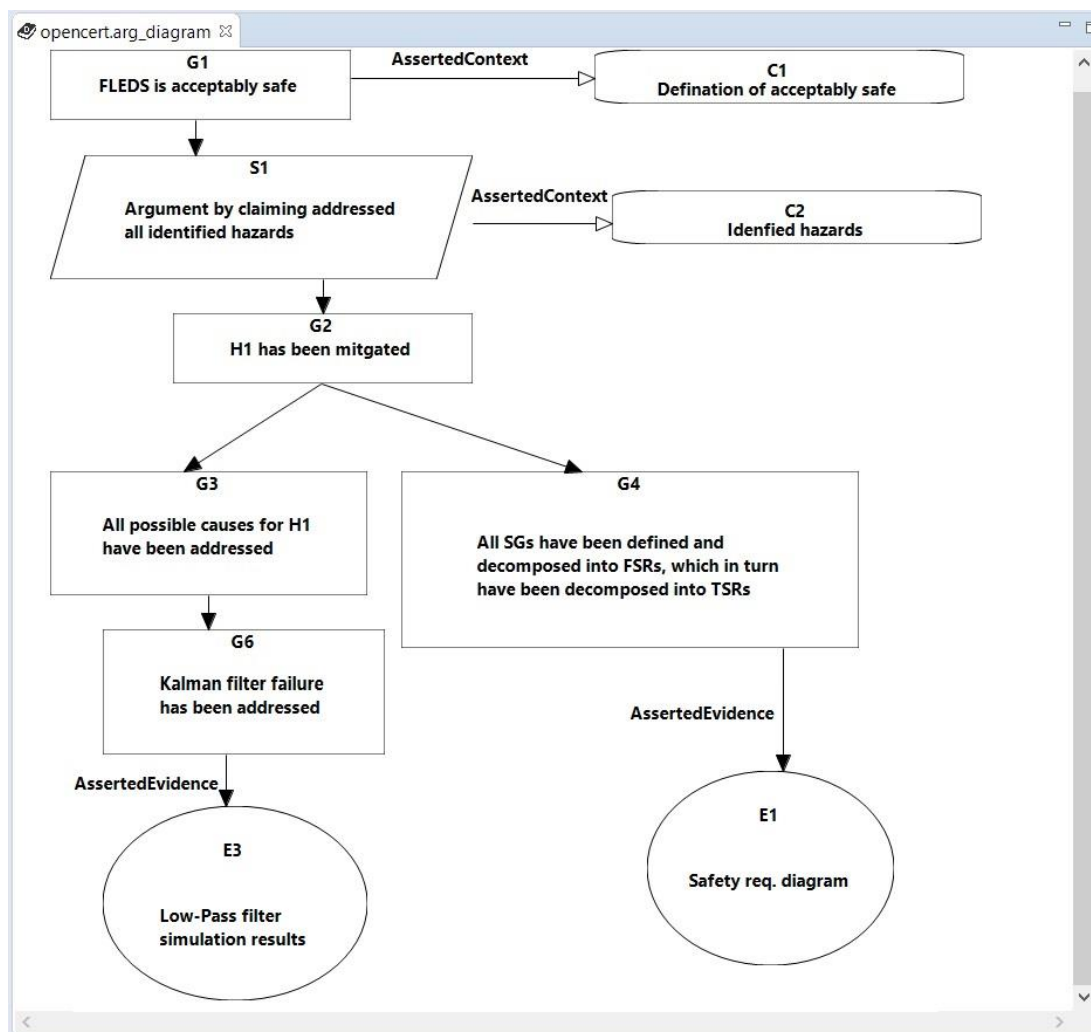


Figure 131. Re-configured argumentation fragment



## Appendix C. Changes with respect to D6.2 (\*)

New Chapters:

Chapter/Appendix	Title
10	AMASS design solution for compliance checking
15	Conclusion
Appendix C	To document the changes.

New Sections:

Section	Title
5.5	Product-related reuse: focus on safety and security analysis artefacts
5.6	Conceptual approach on product reuse
5.7	Model Based Testing for exploring the benefits of re-use of development cycles
5.9.1.1	Types of fallacy
5.9.1.2	Modelling of Safety Processes
5.9.1.3	Detecting Fallacies in Process Models
5.8	Approach on impact analysis and delta analysis based on data indices using Elasticsearch
5.10.1	Argument-fragment generation at the architectural pattern level
8.1.4	Property Specification Patterns for Finite-State Verification
8.1.5	EPF Composer Metamodels
8.1.6	Regorous Metamodels
12.2	Semantic Analysis of Safety Standards
14.2	GQMPS for product-&-assurance case related reuse

Chapters whose number has changed:

Former Chapter No.	New Section No.	Title
11	12	
12	13	
13	14	

Modified Chapters/Sections:

Chapter/Section	Title	Change
1	Introduction	Revised to properly introduce the new deliverable.
2	Recap concerning industrial needs with respect to STO4	Revised w.r.t. changes introduced in D1.1, v.1.2.
4.1.1	Process-related macro and micro (reusable) elements	Better explained the difference between process plan and executed process; Discussed the impact of FAA and RESSAC initiatives on the AMASS solutions.
4.2.1.4	Verification phases of the life-cycle maintenance of SEooC	Revision of the presentation, minor changes.
5.1.5	Definition of an interface for reuse discovery	Revision of the presentation, minor changes.



5.2	Reuse assistance	The main content of the section regarding the design has been re-written.
5.3.4.1	Intra-domain variability management at assurance case level: an automotive assurance case line	The design of this functionality has been completed and exemplified.
5.9	Automatic generation of process-based arguments	Four sub-sections are added within section 5.9.1. Moreover, the table showing the mapping between UMA-compliant process modelling elements, and CACM-compliant argumentation modelling elements is updated.
5.9.2	Generating Process-based Argumentation Representing Executed Processes	This subsection has been revised and completed.
7	AMASS vision for compliance management	Revision of the vision, minor changes.
9.2.2.1	Running Example: ISO 26262	More information related to the standards ISO 26262 was added.
12.1	Representation of Safety Standards with Semantic Technologies Used in Industrial Environments	Revision of the explanation about the enactment of the scenarios proposed to exploit the semantic representation of safety standards.
14.1.2	GQM + Strategies Model for the evaluation of families of processes	Revision and inclusion of parts published in a paper accepted at EuroSPI-2018.
References	References	References have been enriched.
Appendix B	Appendix B	Extended with new images.