ECSEL Research and Innovation actions (RIA)

AMASS

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

AMASS platform validation
D2.9

Work Package:	WP2 Reference Architecture and Integration
Dissemination level:	PU = Public
Status:	Final
Date:	3rd May 2019
Responsible partner:	Morayo Adedjouma/ Bernard Botella (CEA)
Contact information:	{morayo.adedjouma, bernard.botella } AT cea.fr
Document reference:	AMASS_D2.9_WP2_CEA_V1.1

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

Contributors

Names	Organisation
M. Adedjouma, B. Botella, H. Espinoza	Commissariat à L'Energie Atomique et aux Energies Alternatives (CEA)
Zoe Stephenson	Rapita Systems (RPT)
Marc Born	ANSYS (KMT)
Muhammad Atif Javed, Faiz UL Muram, Nils Muellner, Barbara Gallina	Maelardalens Hoegskola (MDH)
Stefano Puri	Intecs (INT)
Alejandra Ruiz, Angel López, Cristina Martinez	TECNALIA Research & Innovation (TEC)

Reviewers

Names	Organisation
Alberto Debiasi (Peer Reviewer)	Fondazione Bruno Kessler (FBK)
Isaac Moreno (Peer Reviewer)	Thales Alenia Space (TAS)
Cristina Martinez (Quality Review)	TECNALIA Research & Innovation (TEC)
Jose Luis de la Vara (TC Review)	Universidad Carlos III de Madrid (UC3)
Barbara Gallina (TC Review)	Maelardalens Hoegskola (MDH)

Document History

Version	Date	Status	Author (Partner)	Remarks
V1.0	2019-02-11	Final version v1.0	B. Botella, M. Adedjouma (CEA)	
V1.1	2019-05-03	Final version v1.1	B. Botella, M. Adedjouma (CEA)	Updating the Sections 4. Tool Qualification and 5. TRL Assessment. Extending the Section 3. AMASS Platform Evaluation with a "Usability analysis"

TABLE OF CONTENTS

List of Figures

List of Tables

Executive Summary

This document (D2.9) is the fourth deliverable of T2.4 “AMASS Platform Validation”. This task aims to integrate the AMASS platform building blocks developed in WP3, 4, 5, and 6, and to validate the developed concepts and technologies in test conditions. The D2.9 deliverable provides an overview of the AMASS open tool platform and accompanied public artefacts. The deliverable includes a summary of the results of the test campaigns that have been executed along the three test development iterations of the Platform, together with an analysis of the coverage of the high-level requirements elicited for the Platform as well as the limitations encountered during validation. Along with these validation campaigns outcomes, the deliverable includes a usability analysis of the AMASS Platform tools.

The deliverable also provides an evaluation of the tool qualification and TRL achieved by (some components of) the platform, and some recommendations concerning the AMASS Platform further usage and evolution as well as some tracks for future exploitations. Appendix A provides the details of the results from the test campaigns. Appendix B includes the latest version of the AMASS Platform Common Assurance & Certification Metamodel (CACM).

1. Introduction

1.1 Scope

AMASS has created and consolidated a de-facto European-wide assurance and certification open tool platform, ecosystem and self-sustainable community spanning the largest Cyber-Physical System vertical markets. Its aim is to lower certification costs in face of rapidly changing product features and market needs. This has been achieved by establishing a novel holistic and reuse-oriented approach for:

- Architecture-driven assurance, fully compatible with standards such as AUTOSAR [17] and Integrated Modular Avionics (IMA) [18].
- Multi-concern assurance, such as compliance demonstration, impact analyses, and compositional assurance of security and safety aspects.
- Seamless interoperability between assurance/certification and engineering activities along with third-party activities (external assessments, supplier assurance).
- Cross/intra-domain re-use of, for instance, semantic standards and product/process assurance.

The AMASS tangible results are:

a) The **AMASS Reference Tool Architecture**, which extends the OPENCOSS [1] and SafeCer [2] conceptual, modelling and methodological frameworks for architecture-driven and multi-concern assurance, as well as for further cross-domain and intra-domain reuse capabilities and seamless interoperability mechanisms (e.g. based on Open Services for Lifecycle Collaboration (OSLC)[1] specifications).

b) The **AMASS Open Tool Platform**, which corresponds to a collaborative tool environment supporting CPS assurance and certification. This Platform represents a concrete implementation of the AMASS Reference Tool Architecture, with a capability for evolution and adaptation, which is released as an open technological solution by the AMASS project. AMASS openness is based on both, standard OSLC Application programming interfaces (APIs) [14] [13] with external tools (e.g. engineering tools including V&V tools) and open-source release of the AMASS building blocks.

c) The **Open AMASS Community**, which manages the project outcomes for maintenance, evolution and industrialization. The Open Community is supported by a governance board, and by rules, policies, and quality models. This includes support for AMASS base tools (tool infrastructure for database and access management, among others) and extension tools (which will enrich the AMASS features). As Eclipse Foundation is part of the AMASS consortium, the PolarSys/Eclipse community [3] has been chosen as the best candidate to host AMASS.

[1] https://open-services.net

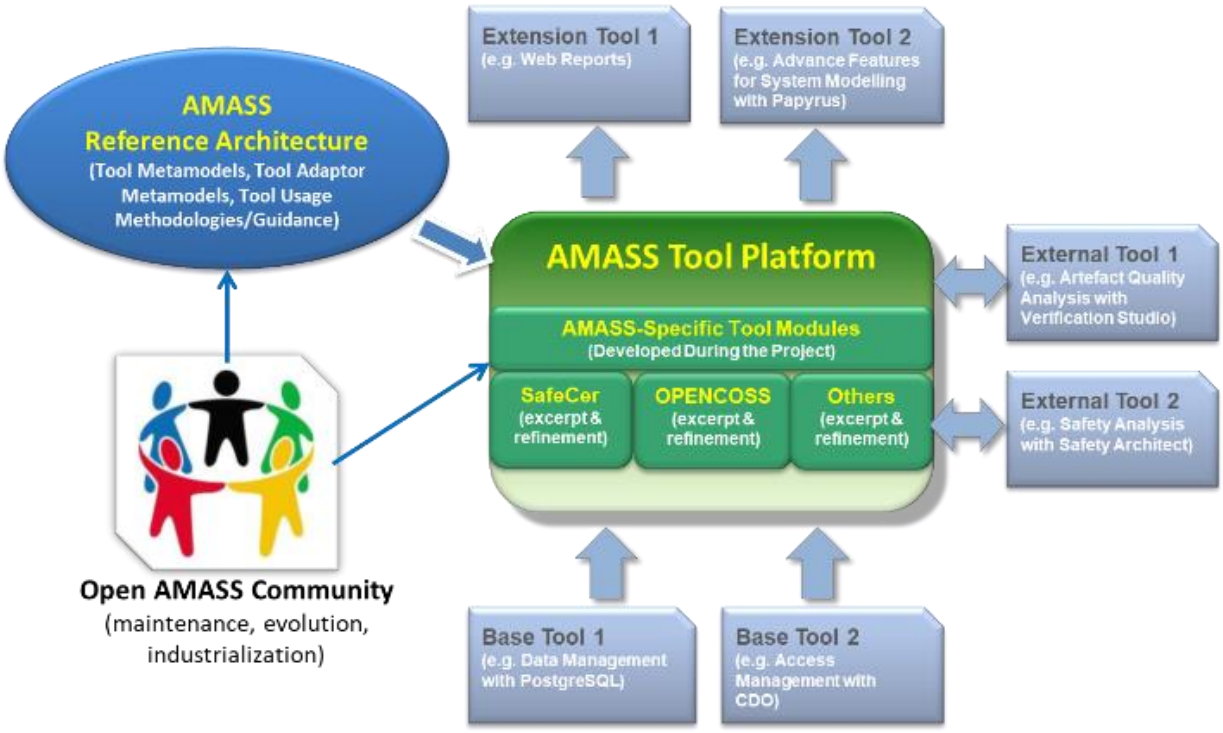


Figure 1. AMASS results

To achieve these results, the AMASS Consortium has decided to follow an incremental approach by developing rapid and early prototypes in three iterations:

1. During the **first prototyping** iteration (Core Prototype), the AMASS Platform Basic Building Blocks were aligned, merged and consolidated at Technology Readiness Level (TRL) 4 (technology validated in laboratory).
2. During the **second prototyping** iteration (Prototype P1), the single AMASS-specific Building Blocks were developed, integrated with previous prototype and benchmarked at TRL 4.
3. Finally, at the **third prototyping** iteration (Prototype P2), all AMASS building blocks have been integrated in a comprehensive toolset operating at TRL 5 (technology validated in a relevant environment).

1.2 Purpose of the Deliverable

This deliverable is the fourth one resulting from Task 2.4 “AMASS Platform Validation”. The purpose of this deliverable is to report the validation activities executed on the AMASS Open Platform. The document provides an overview of the AMASS Platform and a summary of the three validation campaigns (along with the main limitations identified) of this platform based on the high-level requirements that were envisioned at the beginning of the project. Along with these validation campaigns outcomes, the deliverable includes a usability analysis of the AMASS Platform tools. We also present some tool qualification needs according to different standards and evaluate a selected tool chain of the platform regarding them. We provide a TRL evaluation of key components of the AMASS platform and an analysis of some public artefacts associated with the Platform. Finally, some recommendations for the usage and the evolution for the AMASS Platform are provided.

1.3 Relations to other Deliverables

The D2.9 deliverable provides a summary of the previous validation deliverables D2.6 (Integrated AMASS Platform (a)) [38], D2.7 (Integrated AMASS Platform (b)) [20] and D2.8 (Integrated AMASS Platform (c)) [39] about the validation of the AMASS Platform. The deliverable also refers to the D2.1 deliverable (Business

cases and high-level requirements) [8] that defines the business models of the AMASS solutions as well as the requirements to be met by the Platform, and to the D2.4 deliverable [7] (AMASS Reference Architecture (c)), which describes the overall architecture of the AMASS platform including needs from the case studies that must be covered by the Platform.

1.4 Structure of the Document

The deliverable is structured as follows:

- Chapter 2 includes a short presentation of the AMASS Platform, the tooling architecture and technologies used to implement it.
- Chapter 3 presents a summary of the validation campaigns and the usability analysis executed on the Platform.
- Chapter 4 presents a tool qualification evaluation over the selected tool chain of the AMASS Platform with regard to different standards.
- Chapter 5 provides a TRL evaluation of the Platform.
- Chapter 6 presents an analysis of some public artefacts associated with the Platform.
- Chapter 7 presents some recommendations to be considered for further usage and evolution of the AMASS Platform beyond the project.
- Chapter 8 provides some perspectives about the future exploitations of the Platform.
- Chapter 9 concludes the document.
- Appendix A provides a detailed status of the AMASS Platform validation campaigns.
- Appendix B presents an updated version of the CACM.

2. AMASS Platform

Figure 2 provides a high-level picture of the AMASS Reference Tool Architecture (ARTA) where the basic building blocks constituting the Core Prototype are surrounded by a red dash-line and the building blocks implemented in Prototypes P1 and P2 are depicted in green boxes. As part of the overall platform, the AMASS Platform Basic Building Blocks are produced by merging existing technologies from OPENCOSS [1] and SafeCer [2], and other related projects such as CHESS [9]. This core is extended with additional tools in order to address the following concerns: architecture-driven assurance, multi-concern assurance, seamless interoperability and cross/intra-domain reuse. The overall AMASS platform integrates the set of tools in a comprehensive manner and provides the functionalities described in the AMASS deliverable D2.4 [7] (AMASS Reference Architecture (c)).

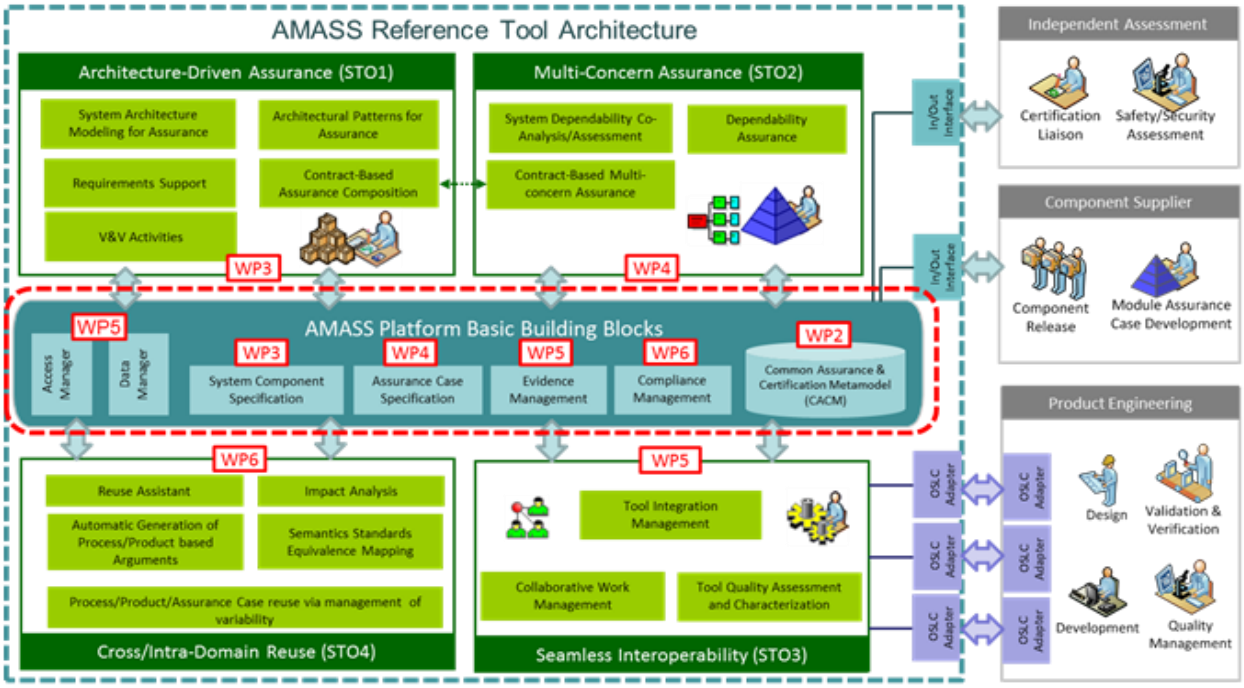


Figure 2. AMASS Reference (High-Level) Architecture (Prototype P2)

2.1 AMASS Platform Tools

The AMASS platform comprises a set of tools which provide the functionalities described in the AMASS deliverable D2.4 [7] (AMASS Reference Architecture (c)). Figure 3 presents an overview of the tool ecosystem composing the platform. The platform has been built upon the following baseline technologies and toolsets:

1. **OpenCert** [1] – AMASS Core edition, which supports Assurance case specification, Dependability Assurance Modelling, and Contract–based multi-concern assurance.
2. **Papyrus** [15] and **CHESS** [9] – AMASS Core edition, which supports Contract modelling, Contract-Based Multi-concern Assurance, and Contract-based trade-off analysis in parameterized architectures with OCRA, NuXmv and XSAP.
3. **EPF-Composer** [19], which supports Assurance process modelling and tailoring to the individual project (the resulting process model is used by WEFACT [11]).
4. **Concerto-FLA** [22], which supports Failure Logic Analysis (FLA) for safety and security-related failure modes.
5. **Capra tool** [16], which supports traceability management.

6. **BVR tool** [10], which supports orthogonal variability management and which, integrated with EPF-Composer supports variability management at process level (enabling process co-assessment as well as cross-concern, cross-domain, and intra-domain reuse of process information), integrated with CHESS supports variability management at product level and, integrated with OpenCert supports variability management at argumentation level.
7. **Open Service for Lifecycle Collaboration** (OSLC) technology and **Knowledge Manager** (KM) toolset [13][14], which support interoperability features.

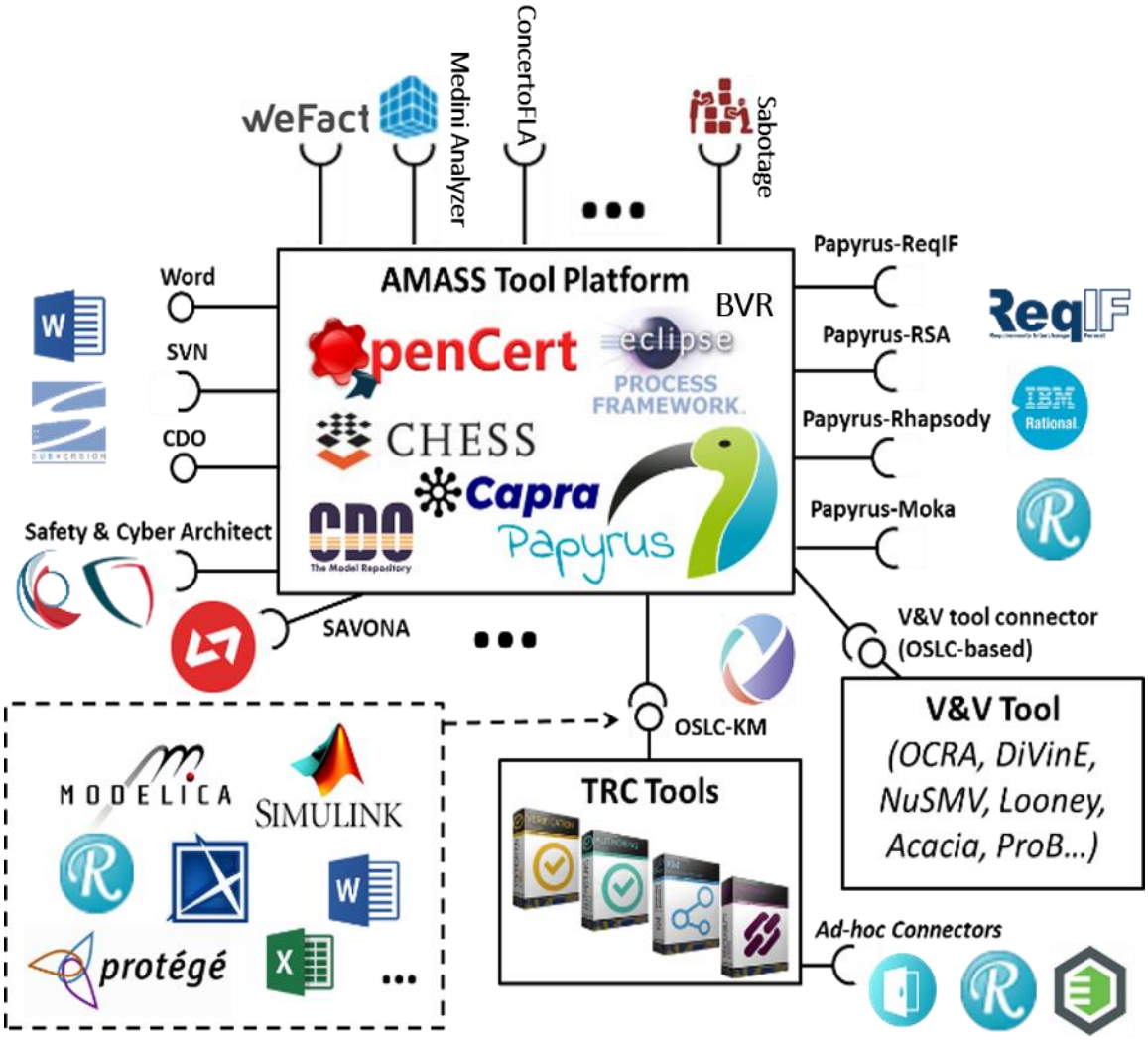


Figure 3. AMASS tool ecosystem

2.2 External Tools connected to the AMASS Platform

The AMASS Platform supports/provides interfaces to several external (and proprietary) tools for further analysis support. The way these tools are connected to AMASS are plural: using seamless interface or by means of import/export features. The main external tools that are connected with the Platform are the following:

1. **WEFACT** [11], which supports the assurance process workflow.
2. **FMVEA** [35], which supports model-based system-dependability co-analysis and –assessment.
3. **MORETO** [35], which supports security analysis and manual or standards-based automated generation of security requirements.

4. **Medini Analyze** [23], which supports the assurance process workflow and allows safety and security analyses.
5. **SAVONA** [27], which supports contract modelling.
6. **Sabotage** [26], which supports fault injection simulation.
7. **SafetyArchitect** [24] and **CyberArchitect** [25], which support dependability co-analysis.
8. **V&V Manager** [36] for verification and validation features.
9. **Verification Studio** [28] for management of requirement analysis, requirements quality analysis and V&V evidence.

Further information on how to use the AMASS features together with these external tools can be found on their respective external tool documentation.

3. AMASS Platform Evaluation

3.1 High level Requirements Coverage

The D2.1 deliverable [8] defines the high-level requirements for the AMASS Platform. The requirements were defined for the basic building blocks composing the AMASS Core Prototype. In addition, some requirements were defined to address the 4 specific project Scientific and Technical Objectives (STOs). In overall, we have obtained the ratio reported in Table 2 for the high-level requirements provided by the Platform versus the requirements elicited in D2.1. The Appendix A details the coverage of the high-level requirements implemented in the AMASS Platform as derived from the validation campaigns.

Table 1. High level requirements coverage

Requirements	Number of supported requirements / Number of requirements elicited
Basic Building Blocks	54/59 (3 Cancelled, 2 Failed)
Architecture-Driven Assurance (STO1)	31/34 (3 cancelled)
Multi-Concern Assurance (STO2)	8/8
Seamless Interoperability (STO3)	36/36
Cross/intra-Domain Reuse (STO4)	12/15 (3 cancelled)

3.2 Summary of the Validation Campaigns

Three validation campaigns were executed on the AMASS Platform. The aim of the campaigns was to validate that the AMASS Platform features satisfy its requirements and to check the system behaviour against user needs and case studies (see D2.4 [7] deliverable). For each feature, we have defined one or several test cases to provide concrete (nominal and some alternative) scenarios about how the AMASS features may be used and when such usage can be regarded as successful. It happens also that the same test case could be valid for several features. Based on the test cases execution, we have defined the validation status of each feature as:

- **Passed**: feature that works as required
- **Passed but**: feature that works but could be enhanced
- **Failed**: feature that does not work
- **Inconclusive**: feature that was not considered in the validation campaign

A rationale has been provided describing why the validation status is “Passed but” or “Failed” during the evaluation of the feature in previous deliverables [20][38][39]. We have also elaborated some tickets for these features within the AMASS Issue-Tracker system in order to report the problems to the Implementation Team for further resolution. The results of the three validation campaigns are summarized in Table 2.

The first validation campaign (Campaign 1) was executed on the AMASS Core Prototype with the aim to test the correct integration of the AMASS Platform Basic Building Blocks and check the correctness of the implementation. We considered 33 features in this campaign, among which 3 features failed.

The second validation campaign (Campaign 2) was executed on the AMASS Prototype P1. The objective of this campaign was to check the resolution of issues related to Prototype Core features and the new functionalities implemented in the Platform. Hence, the 3 features that “Failed” and the feature resulting

with the "Passed but" status in Campaign 1 were included in the Campaign 2. In total, we evaluated 29 features in the campaign, among which 10 features met the user expectations.

The third validation campaign (Campaign 3) was executed on the AMASS Prototype P2 release. This final campaign aimed to test the final AMASS features implementation and their integration based on the reference architecture that was envisioned for the Platform in deliverable D2.4 [7]. In addition to new implemented features, the campaign considered the ones that were not tested and the ones whose test results were found not satisfactory during the previous campaigns. The 9 features that "Failed" and the ones resulting with "Passed but" status (10 features) in Campaign 2 were included in the Campaign 3. Note that these (19) features included some ones from Campaign 1 that still needed to be revised as well. The Campaign 3 included 59 features and only one feature remains failed.

Table 2. Validation campaigns results

Feature results	Campaign 1	Campaign 2	Campaign 3
Passed	29	10	52
Passed but	1	10	6
Failed	3	9	1[2]
Inconclusive	9	14	0
Total	**42**	**43**	**59**

3.3 Usability Analysis

To get feedback on the usability perception of the AMASS Platform, a survey using the System Usability Scale (SUS) questionnaire [61] was performed with the case studies' evaluators. Specifically, eight people participated in the survey.

The SUS is a questionnaire of 10 questions to measure the usability perception, and an indication about the learnability perception in a second dimension – Questions 4 and 10. The participants are given 1–5 scale to fill where 1 means strongly disagree, while 5 means they strongly agree with the statement. In the context of AMASS, we made a questionnaire about the tool chains that the case studies' evaluators were using for their case studies [62]. Figure 4 presents the AMASS SUS questionnaire.

The evaluated tool chains included the following AMASS tools and features: OpenCert, Papyrus/CHESS (including OCRA, NuXmv, xSAP), EPF Composer, BVR Tool, Papyrus SysML, and V&V Manager.

	Strongly Disagree				Strongly Agree
1. I think that I would like to use this tool chain frequently.	1.	2.	3.	4.	5.
2. I found this tool chain unnecessarily complex.	1.	2.	3.	4.	5.
3. I thought this tool chain was easy to use.	1.	2.	3.	4.	5.
4. I think that I would need assistance to be able to use this tool chain.	1.	2.	3.	4.	5.
5. I found the various functions in this tool chain were well integrated.	1.	2.	3.	4.	5.
6. I thought there was too much inconsistency in this tool chain.	1.	2.	3.	4.	5.
7. I would imagine that most people would learn to use this tool chain very quickly.	1.	2.	3.	4.	5.
8. I found this tool chain very cumbersome/ awkward to use.	1.	2.	3.	4.	5.
9. I felt very confident using this tool chain.	1.	2.	3.	4.	5.
10. I needed to learn a lot of things before I could get going with this tool chain.	1.	2.	3.	4.	5.

Figure 4. AMASS System Usability Scale questionnaire

2 Two features that were reported as "Failed" in D2.8 deliverable [39] have been corrected.

To calculate the score of a questionnaire, we have used the following formula:

$$(2.5\ \left(\sum(X_i - 1) + \sum(5 - Y_j)\right))$$

where X_i is the answer to an odd-numbered question and Y_j is the answer to an even-numbered question.

We have interpreted the overall score for the 8 questionnaires using a scale inspired from the SUS score interpretation guideline[3], as shown in Table 3.

Table 3. Usability perception interpretation guideline and results

SUS Score	Grade	AMASS Rating	CS tool chain evaluation
> 80.3	A	Excellent user experience	CS7:US1-2
68-80.3	B	Good user experience	
68	C	Fair usability	
51-68	D	Tolerable usability	CS6, CS5, CS10
< 51	E	Potential for usability optimisation	CS1, CS4, CS7:US3, CS11

The case studies evaluators perceived the easy-to-use of their tool chains, as follows:

- CS7:US1-2 evaluated the usability of the tool chain comprised of Papyrus SysML and V&V Manager as an "*Excellent user experience*",
- The usability of the OpenCert tool was evaluated as *Tolerable* by CS6 while CS1 identified a "*Potential for usability optimisation*".
- CS5 evaluated the Papyrus/CHESS usability as *Tolerable*.
- CS10 evaluated the tool chain comprised of Papyrus CHESS, OpenCert and RQT as having a *Tolerable* usability.
- CS4 evaluated the tool chain that includes Papyrus CHESS and OpenCert, as having "*Potential for usability optimisation*".
- CS7:US3 evaluated the usability of the tool chain comprising EPF Composer, BVR tool and OpenCert tool as having "*Potential for usability optimisation*".
- CS11 identified some "*Potential for usability optimisation*" for the tool chain that includes Papyrus CHESS, EPF Composer, BVR tool and OpenCert tool.

In order to analyse of the usability perception provided by the case studies' evaluators, it should be noted that the AMASS platform is a TRL5 prototype tool that needs to be enhanced in order to achieve a higher readiness level that could fit an industrial deployment. That explains why some participants that have evaluated the platform as a ready-to-use commercial solution to be used in a daily basis in their companies (e.g. CS10, CS7:US3, CS6) have provided lower usability ratings than other participants (e.g. CS7:US1-2) that have evaluated positively the tool potential outcome of an RIA project such as AMASS.

Regarding the learnability perception assessment, the Questions 4 and 10 in the SUS questionnaire provide an indication about the learnability perception (see Figure 4). The answers to these two questions for each of the 8 participants in the analysis are shown in the Table 4.

These answers (based on the 1-5 scale as of the SUS questionnaire) indicate that most of the participants will need help to use the tool chain, independently of their background knowledge of the concepts and technologies used within that tool chain.

[3] Usability perception interpretation guideline: https://uxplanet.org/how-to-measure-product-usability-with-the-system-usability-scale-sus-score-69f3875b858f

Table 4. Learnability perception results

Score	**Question 4:** I think that I would need assistance to be able to use this tool chain	**Question 10:** I needed to learn a lot of things before I could get going with this tool chain
1 (Strongly Disagree)	★	★★
2		★★
3	★	★
4	★★★★	★
5 (Strongly Agree)	★★	★★

3.4 Analysis of the Results

From the test campaigns, some findings that are worth to mention have been identified.

Platform validation document baseline

There arises difficulty to have consistent and up to date reference documents to base the validation of the platform on. The D2.1 deliverable [8], the D2.4 deliverable [7] and the User Manual [5], which are the main input documents for the validation presented some discrepancies.

Some tests have been based on requirements and use case scenarios that were either incomplete or no more up to date, making it difficult to exploit. We also found some discrepancies between the User Manual [5] and Developers' Guide [6] and the software. These discrepancies were due to the timeline of the validation that took place just after the implementation of the platform while the developers were still in the process of updating the reference documents. A validation goal was also to help identify the inconsistencies in the documentation, so the developers could make updates accordingly.

- About the Usage Scenarios

  The description of some usage scenarios did not match the user expectations for some requirements that were associated with them, or the steps provided for the scenarios were so generic that they did not help understand the expected results. We also missed several usage scenarios to understand the user needs behind some high-level requirements, so it was difficult to define accurate test scenarios to validate them, determining a difference between the test definition and the actual implementation. In these cases, both the validation team and the implementation team worked closely during interviews to overcome these issues.

- About the User Manual

  The AMASS User Manual has evolved during the several iterations of the Platform (Core, P1, P2), as result some guidelines were either incomplete or no more up to date. For example, some screenshots created during the first iteration did not correspond exactly to the final GUI of the Platform, despite the contents were very similar. When the validation feedback requested the change, the modifications were done.

  Because of the collaborative definition of the document, the level of description of different features varied, some were are pretty much well described while others were described in a too abstract manner (with some implicit information embedded sometimes). This is one of the challenges shared by many open source communities where different developers from different companies and backgrounds provide their contributions.

  We have also identified that there exist some minor functionalities provided by the Platform whose potential usage description is not included in the User Manual.

Cancelled Implementation Features

The AMASS Platform, in the final stage of development, does not implement some features that were defined in D2.1 [8], most of these features were related to the basic building blocks of the Platform. Table 5 lists the features cancelled with the MosCow[4] priority that was defined for them in the D2.1 deliverable [8]. It should be noted that only one feature was mandatory while the others were only desirable.

The "WP4_ACS_009 Find high level claims" feature, which was initially considered as mandatory, was later on deprecated and no implementation was provided since it could be easily supported by a visual review. In general, the cancellation was motivated by the feedback of some Case Study owners, being not so convinced by the expected level of automation of the features to respond to their concerns.

Table 5. Cancelled AMASS features

Feature ID	Description	MosCow Priority
WP3_APL_004	Architectural Patterns suggestions	Could
WP3_VVA_008	Automatic test cases specification from assurance requirements specification	Shall
WP3_VVA_009	Capability to connect to tools for test case generation based on assurance requirements specification of a component/system	Shall
WP3_SC_003	Modelling languages for component model	Could
WP4_ACS_009	Find high level claims	Must
P4_ACS_012	Formal validation of assumptions and context when arguments modules are connected	Could
WP6_RA_007	Provision of metrics about process-related reuse (e.g., size of commonality)	Could
WP6_RA_008	Provision of metrics about product-related reuse (e.g., size of commonality)	Could
WP6_RA_009	Provision of metrics about assurance case-related reuse (e.g., size of commonality)	Could

Platform Usability

The Platform provides several menus for the features distributed over the GUI. The same feature can appear in the toolbar, the model menu and the context menu. Some other features provide a menu in a unique place, there is no rationale to explain these choices. The plurality of the menus can be confusing for the user. As it is reported above, there exist some features within the Platform with dedicated menus but with no description in the user manual about their usage. There also exist menus that seem not activated, i.e. not linked to any feature or are only activated in certain circumstances and the test cases have not been able to reproduce then and cover them. This diversity in the GUI is commonly observed in open source community where different people from different background develop on the same platform.

The usability perception analysis made by the case studies evaluators let confirm these technical usability issues. Nevertheless, it must be noted that the AMASS platform involves TRL5 technology outcome of a RIA project, so it should be considered as a prototype tool to be enhanced in order to achieve a higher readiness level that could fit an industrial deployment.

[4] Must have, Should have, Could have, and Won't have but would like

External tools

Some high-level requirements elicited in D2.1 [8] have been implemented only by external proprietary tools. For those external tools that do not provide seamless interoperability means with the Platform, it was necessary to get resources and information from the tool providers (licence, executable, installation and user manual). The process for collecting the information for the validation purpose was not well defined and it has impacted somehow the validation campaigns timeline.

4. Tool Qualification

4.1 Qualification Concept and Needs

Tool qualification is the process of assessing the trustworthiness of a software tool that is used in the context of a safety related application or system. Safety standards and guidance such as IEC 61508 [32], ED-215 (DO-330) [33] or ISO-26262 [34] call for tool qualification.

4.1.1 Tool qualification according to IEC 61508

In the current version of IEC 61508 [32], the requirements for tool qualification are rather weakly specified and distributed through the standard. (There is a rework of this ongoing)

The current version of IEC 61508 provides levels for "software off-line support tools" (i.e. tools used at development time and not at run-time of the safety critical system):

- **T1**: the tool does not generate outputs that can directly or indirectly contribute to the executable code of the safety related system.
- **T2**: the tool supports testing or verification activities; errors in the tool can lead to undetected defects in the executable software, but the tool cannot introduce those defects.
- **T3**: the tool generates outputs that directly or indirectly contribute to the executable code.

Based on these levels, requirements for further qualification activities are provided. Only for T2 and T3 tools such requirements exist. For **T2** and **T3** tools, it is necessary to assess the usage of the tool and the effect that potential errors of the tool could have on the executable software. Further, mitigation measures must be planned and executed. For **T3** tools, it is additionally necessary to provide evidence that the tool behaviour conforms to its specification. Tool validation may be appropriate to demonstrate such evidence.

4.1.2 Tool qualification according to ED-215 (DO-330)

For the Aerospace domain, Document ED-215 (DO-330) [33] gives general guidance for tool qualification based on a Tool Qualification Level (TQL) ranging from 1 to 5. Typically, TQL 5 is assigned for tools that *assess* software development artefacts, and the other TQLs are assigned for tools that *create* software development artefacts.

4.1.3 Tool qualification according to ISO 26262

For the Automotive domain, the tool qualification requirements are specified in ISO 26262-8 section 11 [34]. Generally, to use software tools in safety related automotive product development, confidence is required that these tools achieve the following goals (citation of ISO 26262-8, clause 11.2):

1. the risk of systematic faults in the developed product due to malfunctions of the software tool leading to erroneous outputs is minimized,
2. the development process is adequate with respect to compliance with ISO 26262, if activities or tasks required by ISO 26262 rely on the correct functioning of the used software tool.

This confidence is given by a so-called **Tool Confidence level** (**TCL**). Basically, the TCL needs to be determined along two criteria:

1. **Tool Impact (TI)**: the possibility that malfunctions of the tool and its corresponding output can introduce or fail to detect errors in a safety-related item or element being developed. Possible values are: TI1 – *no impact possibility*, and TI2 – *all other cases*.
2. **Tool Error Detection (TD)**: the confidence in preventing or detecting errors in the output of the tool. Possible values for the degree that erroneous output is prevented or detected: TD1 – *high degree*, TD2 – *medium degree*, and TD3 – *all other cases*.

Both criteria are then combined into a classification chart as visualized in the Figure 5.

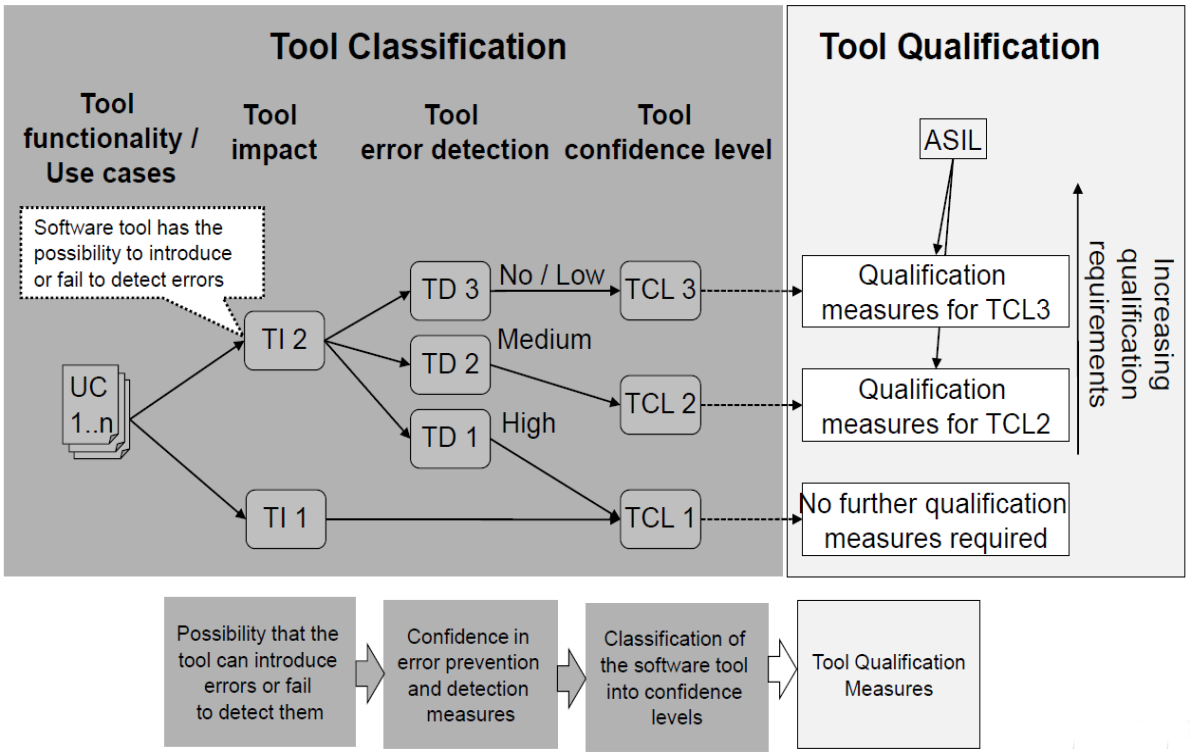


Figure 5. Determination of the tool confidence level (TCL)

As indicated, tool qualification always starts with the *use cases* of a tool. This is noteworthy, since the standard realizes the fact that generic tools like MS Excel are sometimes used for various different purposes, ranging from requirements management to reliability computations (e.g. hardware metrics). Finally, the actual tool qualification *measures* are only required for TCL2 and TCL3 and derived depending on the ASIL of the system being developed as stated in the following table (cp. clause 11.4.6 of ISO 26262-8).

Table 6. Methods for the qualification of tools according to ISO 26262

Methods	TCL 2				TCL 3			
	ASIL A	ASIL B	ASIL C	ASIL D	ASIL A	ASIL B	ASIL C	ASIL D
1a Increased confidence from use	++	++	++	+	++	++	+	+
1b Evaluation of the tool development process	++	++	++	+	++	++	+	+
1c Validation of the software tool	+	+	+	++	+	+	++	++
1d Development in accordance with a safety standard	+	+	+	++	+	+	++	++

If the first option "*Increased confidence from use*" is applied, a rationale for confidence must be given based on its usage history and a number of constraints (e.g. specification of tool is unchanged, tool defects are systematically tracked and resolved, etc.).

The second option "*Evaluation of the tool development process*" is effectively an assessment based on an appropriate national or international standard.

If at least the specification of a tool is available, option 1c "*Validation of the software tool*" is an alternative to the full assessment that consists of a validation of the tool behaviour and its outputs to define counter measures for avoidance or detection of any issues.

The last option 1d "*Development in accordance with a safety standard*" is in practice not relevant, since the development of software tools is almost never following the strict rules defined by any safety standard, due to reasons of the complexity of tools, development processes, and resources/cost considerations at vendor companies.

In addition to the ISO 26262 standard, the Association of European Suppliers for Automotive Software (AESAS) has published recommendations with respect to tool qualification. The overall recommendation is to *avoid the need for qualification of complex tools*. This can be achieved by realizing critical functionalities in small and simple tools, which can then be TCL2 or TCL3 qualified (depending on the ASIL). Moreover, the development of ASIL products should not be dependent on complex tools – or in other words: complex tools should only be TCL1. AESAS sees two means to achieve this:

1. Tool generated artefacts shall be included in product tests. This can be done e.g. for any piece of software binary that is going through a verification procedure.
2. After product tests, modifications like configuration must be verified (e.g. by manual inspection or product's internal plausibility checking).

Figure 6 shows how the tool qualification requirements and process in the ISO 26262 are captured in a reference framework by the OpenCert tool.

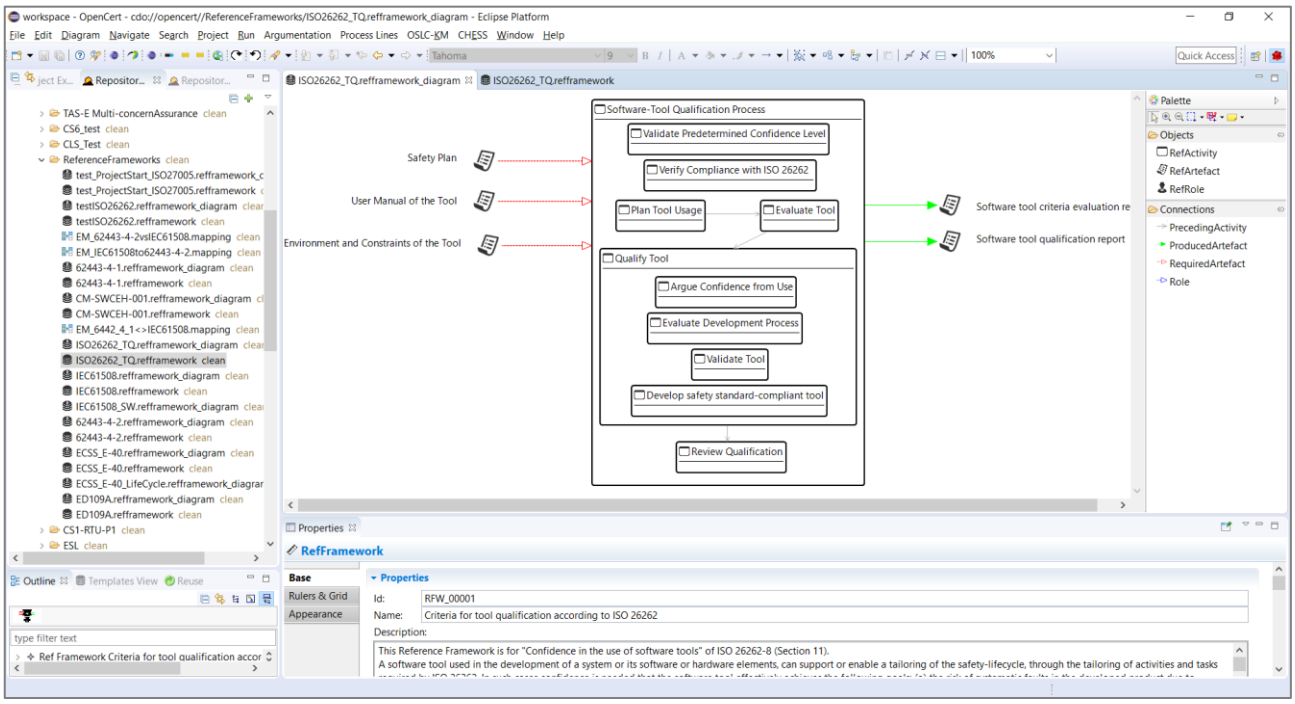


Figure 6. Modelling of tool qualification requirements according to ISO 26262 using OpenCert reference framework

The AESAS consortium gives also some examples, including code generators and compilers, how to implement the guidelines. The work of AESAS is continued by ZVEI (Zentralverband der Elektroindustrie). For further details please refer to www.zvei.org.

In summary, ISO 26262 demands measures to establish confidence in software tools, but the formulation leaves freedom to the industry to find a suitable way to apply it in a dedicated development process. The automotive companies (and not the tool vendors) are responsible for establishing confidence in the tools.

4.1.4 Summary of the tool qualification needs

All the standards mentioned above follow a similar approach: first, the tool must be evaluated/classified taking into account the intended tool usage, the potential impact of errors and the capabilities to detect such errors. Typically, levels are provided (e.g. TCL in ISO 26262, TQL in ED-215 (DO-330) or TL in IEC 61508). Based on these levels, tools can be just used, or they must be qualified according to the requirements of the applicable standard.

For a simple set of tools this is reasonably easy to determine, but for a diverse collection of interacting tools such as in the AMASS Platform Tools ecosystem (see Figure 7), a dedicated *tool chain analysis* is needed.

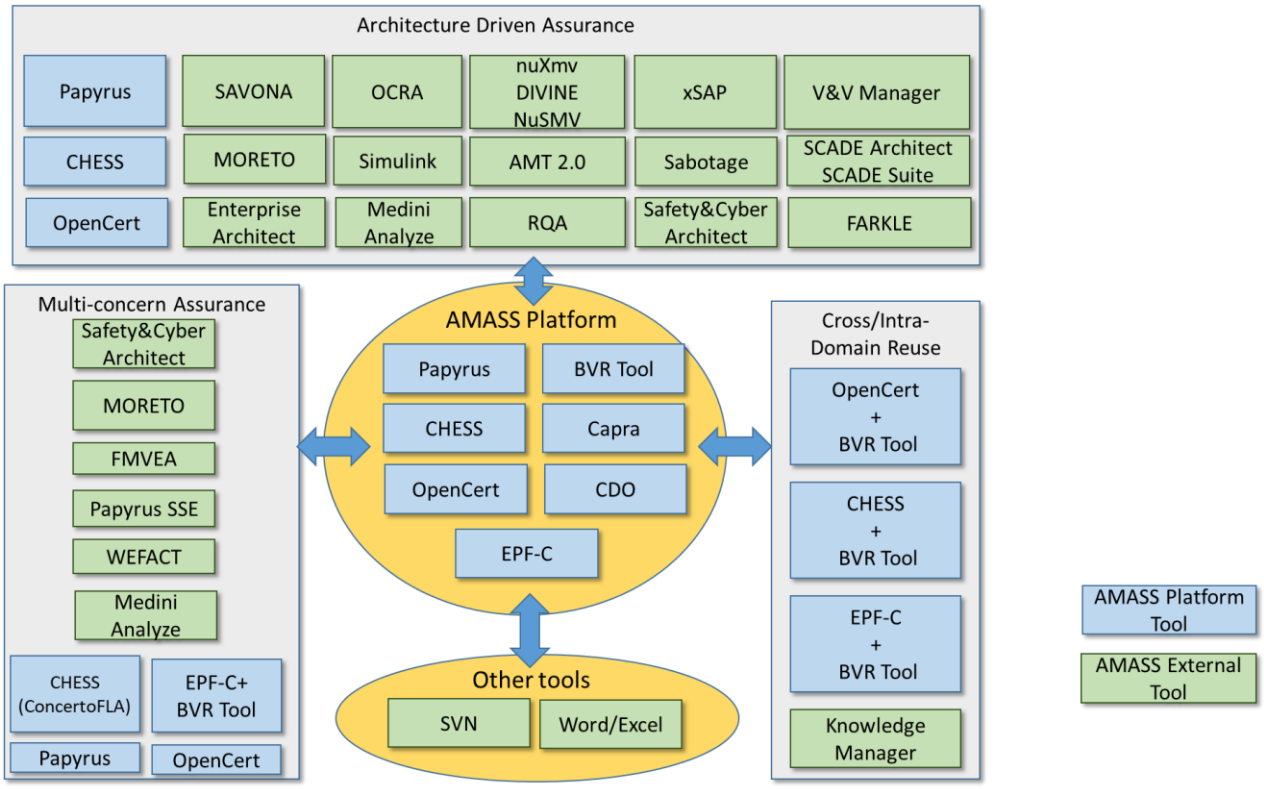


Figure 7. AMASS Platform Tools ecosystem

In the following sections, we introduce the AMASS *tool chain analysis* process, apply it to the AMASS Platform core tools to provide the evaluation based on tool qualification criteria from standards, and then discuss some of the implications of its findings on the tools. Finally, we summarise risk areas that assessors are likely to examine when discussing the qualification of such tools.

4.2 Tool Chain Analysis Process

To perform the tool chain analysis, we specify for each tool:

- Where it obtains its input
- Where it produces its output
- What response it has to possible failures modes of the input
- What additional failure modes it can contribute to the output
- What additional effects the underlying computation platform and network infrastructure can have on the output

Then, by following the chains of responses and contributions, we can determine whether any of the tools can contribute to an undesirable outcome in the overall tool chain output. Having found these, we can

suggest mitigations that become derived requirements placed on those tools that are eligible for qualification. This then forces qualification of those tools.

Tool chain assessment of this kind is a highly specialised form of failure logic analysis (FLA), specifically Failure Propagation Transform Calculus (FPTC). However, to use a tool set including CHESS-FLA or Concerto-FLA to reason about itself invites disaster, if later one finds that a defect in one of those tools masked its own reasoning about that very defect. To help avoid this, we have started this analysis without reference to specific failure logics, so that it can be more easily adapted to an alternative tool approach. This also helps to uncouple the analysis from several issues relating to using the technology in such a different domain:

- In system safety analysis we are concerned with specific hazards and the measures taken to address the risk associated with those hazards. In tool chain analysis, we are concerned with classes of tool failure and measures available within the tool architecture to prevent or detect those classes of failure.
- In many implementations of FPTC it is assumed that the general-purpose failure modes (*early, late, commission, omission, coarse value, fine value*) derived from applying HAZard and OPerability analysis (HAZOP) guidewords to software systems will be suitable for everything analysed. When discussion on software tools, and especially those that operate on a persistent model, these failure modes are no longer appropriate.

To address these concerns fully is beyond the scope of this work. However, we expect to be able to use the following **general classes of tool failure**:

- *Functional change* of the generated system with respect to the behaviour intended of the input
- *Violation of criticality protection* of the generated system
- *Insufficient testing* of the generated system
- *False positive* in the assessment of the generated output, claiming that it complies with its requirements or some other assessment criteria, when it does not

Note that the above classes apply regardless of whether the output is automatically generated (e.g. from Simulink) or manually generated from the low-level specifications in the model. In practice, it is a combination of both.

4.3 Tool Chain Analysis Results

We have identified some functionalities of the AMASS Platform tool set (Figure 2) that are candidates for analysis. Below are these areas highlighted in **bold**, together with their supporting tool:

- Architecture Driven Assurance (STO1)
  - **System Component Specification** **(CHESS Plugin for Papyrus)**
  - **System Architecture Modelling for Assurance** **(CHESS Plugin for Papyrus)**
  - Architectural Patterns for Assurance
  - **Contract-Based Assurance Composition** **(CHESS Plugin for Papyrus)**
  - Requirements Support
  - **Traceability support** **(CHESS Plugin for Papyrus, Capra)**
  - **V&V Activities** **(CHESS Plugin for Papyrus, Concerto-FLA)**
- Multi-Concern Assurance (STO2)
  - Assurance Case Specification
  - **Dependability Assurance** **(OpenCert, CDO)**
  - **System Dependability Co-Analysis/Co-Assessment** **(CHESS Plugin for Papyrus, BVR with EPF-Composer)**
  - **Contract-based Multi-concern assurance** **(CHESS Plugin for Papyrus)**

- Seamless Interoperability (STO3)
  - Evidence Management
  - Tool Integration Management
  - Collaborative Work Management
  - Tool Quality Assessment and Characterisation
- Cross/Intra-Domain Reuse (STO4)
  - **Compliance Management (OpenCert, CDO)**
  - **Reuse Assistant (OpenCert, CDO)**
  - Semantics Standards Equivalence Mapping
  - Impact Analysis
  - **Process-related reuse via management of variability at process level (EPF Composer, BVR)**
  - **Product-related reuse via management of variability at product level (CHESS Plugin for Papyrus)**
  - **Assurance Case-related reuse via management of variability at assurance level (BVR, OpenCert)**
  - Automatic generation of process-based arguments
  - Automatic generation of product-based arguments

Table 7 presents the qualification results for some of the AMASS supporting tools according to the tool qualification criteria in IEC 61508, ISO 26262 and ED-215 standards and following the tool chain analysis process described in Section 4.2.

Table 7. Qualification results of the AMASS Platform tools

Tool	Tool qualification criteria		
	IEC 61508	ISO 26262	ED-215 (DO-330)
OpenCert	T1	TCL1	TQL5
Papyrus & CHESS	T2	TCL2	TQL5
EPF-Composer	T1	TCL1	TQL5
Concerto-FLA	T1	TCL2	TQL5
Capra	T1	TCL1	TQL5
BVR	T2	TCL2	TQL1

In our findings, the supporting tools of most interest are the CHESS Plugin for Papyrus, the BVR system, the CDO store and the OpenCert tool as we detail below.

4.3.1 Tool 1: CHESS Plugin for Papyrus

The CHESS plugin for Papyrus augments Papyrus models with dependability information, initiates analyses of the model, and stores results from the analyses.

- If the model contains a functional change error with respect to its intended behaviour, analysis performed on the model is not guaranteed to discover this error.
- A functional change error in the model with respect to its intended behaviour, left unchallenged, results in a functional change in the generated system. Note that it is more likely that the functional change error is left unchallenged if it is only tested by tests generated from the same model.
- The plugin provides criticality validation checks. If these checks are not implemented correctly, this could miss a criticality constraint, and the resulting generated code could have insufficient testing for its pre-determined criticality level or a violation of criticality protection with respect to the overall disposition of mixed criticalities.

- The plugin provides interface checks. If these checks are not implemented correctly, they could miss an incompatible data type or direction that the compiler accepts, leading to a functional change.
- The plugin provides deployment checks. If these checks are not implemented correctly, they could miss an ambiguous allocation that a later stage then allocates to a core or partition other than intended, which could lead to a violation of criticality protection.
- The plugin provides schedulability analysis, which could fail and claim that the system is schedulable when it is not (e.g. from an error in the algorithm, using an out of date algorithm, or bad assumptions about the OS behaviour). This could lead to a false positive result, and an undetected violation of criticality protection or a functional change.
- The plugin provides dependability analysis, which could claim false positive freedom from failure leading to undetected violation of criticality protection or functional change.
- The plugin provides contract verification, which could claim false positive contract compliance for a system that violates its contract. This could lead to undetected functional change or violation of criticality protection.

4.3.2 Tool 2: BVR tool

The BVR tool provides feature modelling, feature resolution and feature realization. This involves coordinated scripted changes that customise the model so that it's appropriate for some intended use.

- If the model itself misses information about a feature, that feature will always be present in every instance, but there will be evidence that anybody explicitly decided that it should be present, and as a result it could be that some instance is constructed with a functional change with respect to what was intended.
- If the model has too many selections, or the selections are too tightly constrained, then a feature could end up missing, resulting in some instance that contains a functional change with respect to what was intended.
- If a realisation is incorrectly specified, it could lead to an unintended model result and a functional change.
- If a realisation fails to update test artefacts correctly with respect to development artefacts it could result in insufficiently tested software.
- If the BVR tool itself fails to spot incompatible resolutions, or executes realizations incorrectly, it could end up with any of the above scenarios: functional change or insufficiently tested software.

4.3.3 Tool 3: OpenCert tool and CDO store

OpenCert tool aims to provide support for assurance compliance. It does not introduce errors into the design or code of the system, however it can introduce errors in the standards compliance process by introducing errors in the evidences stored for compliance purposes or in the compliance justification. The OpenCert tool relies on CDO technology to store the data related to standards compliance accountability. So, a specific analysis on CDO Data storage is proposed in section 4.3.3.1.

- The OpenCert tool has become the open platform where other tools connect and provide information such as BVR or CHESS. If OpenCert is not use standalone, but connecting to these tools, then the previous analysis should be taken into account, as it could spread the introduced errors.
- OpenCert provides accountability functionality for standards compliance management. If the mappings between the standard requirements and the evidences provided are done incorrectly by the user, it could produce a false feeling of compliance when it is not the case. However, it is always requested to go through a manual assessment by a qualified assessor that will manually

supervise not only the compliance arguments but also the content of the evidences, discovering in this step any error that could have been introduced.

- When OpenCert is used to provide suggestions of the possible evidences to be reused, it could introduce undetected errors, either by nor reusing the right evidence or by introducing errors in the content of that reused evidence. Similarly, to previously mentioned, the final step is the assessment by the qualified assessor. In this step the assessor should identify the remaining errors.

4.3.3.1 CDO data store

The CDO data store provides model storage and persistence for Eclipse tools.

As a platform, the possible failures of the CDO store must be modelled as inputs affecting all the other tools. For example:

- The store could become corrupted, leading to an undetected functional change.
- The store could be valid but access to the store does not retrieve the same entity as was placed in the store, leading to an undetected functional change. Note that this could arise in a systematic way due to an unfortunate interaction between different data conversions, as commonly found when dealing with data interchange.
- The store could fail to update when changes are requested, leaving part of the model in a stale state with respect to the rest of the model. Depending on the nature of the entities involved, this could lead to an undetected functional change or insufficiently tested software.

4.3.4 Identified Risk Areas

As part of the assessment we noted the following further risk areas:

- We only assessed safety issues - the propagation of unintentional failures. The system is also prone to deliberate attacks and should be secured from such. For example, there could be a defect in the tools allowing a rogue agent to insert a backdoor into the generated system and hide it from the user and the analysis, while leaving the backdoor available to code generation.
- Several failure flows relate to “garbage in, garbage out” scenarios. That is, where the tools accept a complex data structure such as a model as input, the user must also provide for validation of that input. Some tool features (e.g. fault tree slicing, requirements animation) can assist this process but it is ultimately outside of the domain of the tools themselves.
- Several failure flows that result in large collections of failure modes relate to stale data, and particularly undetectable failures to re-import updated analysis results. Some attention should be given, architecturally, to systematising all re-import behaviours and providing extensive support for testing these areas during tool qualification.
- The platform effect is difficult to model and difficult to reason about. Since everything revolves around the CHESS model and the CDO store, we would expect to have to qualify those as part of the overall tool offering. They would benefit from dedicated tool qualification.
- We only checked the design as it stands, not the validation of the design. For example, there are a number of integrity checks in the CHESS analysis, but there is nothing in our analysis that shows that this is a sufficiently complete set of checks to assure the integrity of the corresponding implemented code.
- We assessed the tools mainly from the point of view of an ED-12C Criteria 3 tool (ED-215 TQL 5). However, if the CHESS model is used as an input to any kind of code generation technology, then:
  - The code generation technology needs to be qualified at TQL 1 to be usable in all critical applications.
  - The tool user has a responsibility to validate the input to the code generation (e.g. through requirements-based design, test, etc.).

- If the tool user cannot realistically see and work with the same input that is sent to the code generation technology, then the tools that sit between the artefacts that the tool user uses, and the artefacts that the code generator uses, should also be qualified at TQL 1 to address this gap.

5. TRL Assessment

5.1 TRL Definition

The design of an innovative system depends on the evolution of technical knowledge, "The development of new functionalities of a system typically depends on a previous successful advanced technology research and development efforts" [29]. The assessment of a new technology by the organizations allows risks mitigation in a project, assisting the project manager in prioritizing resources for the development of critical technologies that prove to be immature at an early stage of the project [31]. To assess the maturity of a technology the TRL (Technology Readiness Level) methodology was developed by NASA.

The TRL methodology currently defined in ISO 16290:2014 standard [30] consists of a rating of nine levels shown in Figure 8. The evaluation is done through a list of requirements that qualify technology to the next level, the level assigned to technology is the highest level that has the requirements met [29]. This methodology is widely accepted and applied and spread to the most diverse branches of developed economies.

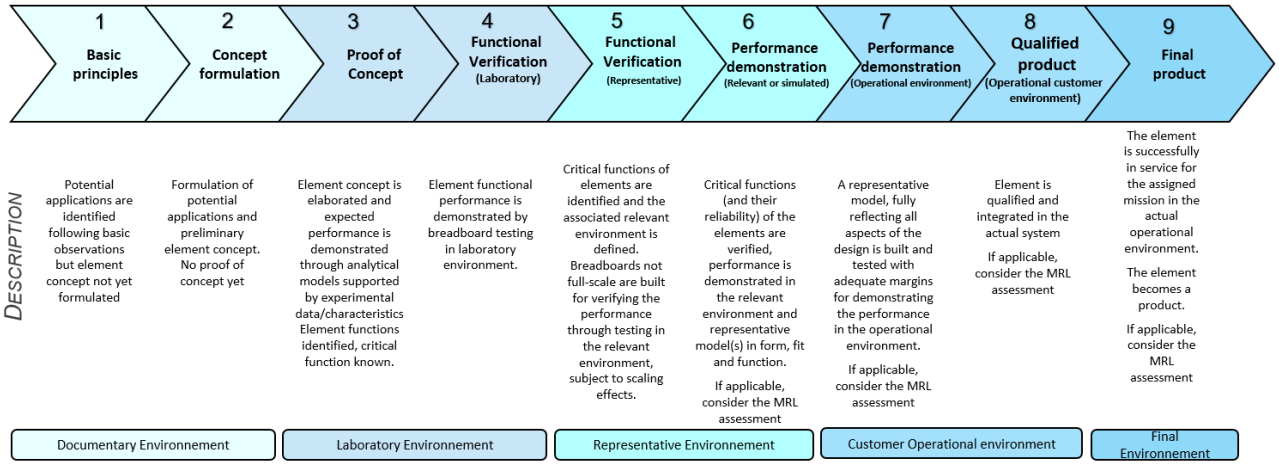


Figure 8. TRL methodology

We have used the TRL methodology to assess some core components of the AMASS Tools platform (see Figure 7), namely the **Papyrus modelling tool**, the **CHESS plugin for Papyrus**, the **OpenCert tool**, and the integration of the **EPF Composer and the BVR tool** with other AMASS tools. In overall, the development done in the AMASS Platform to enhance these components with improved and new functionalities has achieved a maturity level between **TRL 4** and **TRL 5**.

5.2 TRL Assessment: Papyrus Modelling

5.2.1 Papyrus Modelling

The AMASS platform has reused some features of the Papyrus platform to build new features.

The particular features of the Papyrus platform concerned are the following:

- Papyrus modelling capabilities and languages support including UML, SysML and MARTE profiles and associated UI (diagrams, tables, properties view, model explorer, etc.).
- Papyrus DSML and UML profiles definition.
- Papyrus customization mechanism enabling to construct a DSL editor using any notation (text, symbols, style sheets) with corresponding perspectives.
- Requirements extending related SysML requirements profiles and diagrams features.

- Papyrus integration with others modelling tools, e.g. Rhapsody.

These features have been used in several projects and deployed in some companies [40][41][42]. The feedback from their usage in these projects and companies indicates that the tool successfully worked as expected in a representative environment and hence it achieves a level of maturity of **TRL 7**, as justified in Table 8.

Table 8. AMASS Core Papyrus TRL assessment

Level	Supporting information	Status	Justifications
TRL 1: Basic Principles	Published research that identifies the principles that underlie this technology. References to who, where, when.	Passed	The principles behind Papyrus platform has been reported in papers, e.g.: *Lanusse A., Gérard S., Terrier F. (1999) Real-Time Modeling with* UML*:* The *ACCORD* Approach*. In: Bézivin J., Muller PA. (eds). UML 1998. LNCS, vol 1618. Springer* *Gérard S., Terrier F., Voros N.S., Koulamas C. (2001) Efficient System Modeling of Complex Real-Time Industrial Networks Using the* ACCORD/UML *Methodology. DIPES 2000. IFIP, vol 61. Springer, Boston, MA*
TRL 2: Concept formulation	Publications or other references that outline the application being considered and that provide analysis to support the concept.	Passed	Papyrus offers an unhampered access to the full extent of standardized modelling languages like UML, SysML to support the model-based approach in general, as reported in papers, e.g.: *Gérard S., Terrier F., Tanguy Y. (2002) Using the Model Paradigm for Real-Time Systems Development: ACCORD/UML. OOIS 2002. LNCS, vol 2426. Springer* *Phan T.H., Gerard S., Terrier F. (2004) Real-Time System Modeling with ACCORD/UML Methodology. In: Grimm C. (eds) Languages for System Specification. Springer,* *Mraidha C., Robert S., Gérard S., Servat D. (2004) MDA Platform for Complex Embedded Systems Development. DIPES 2004, vol 150. Springer*

Level	Supporting information	Status	Justifications
TRL 3: Proof of concept	Results of laboratory tests performed to measure parameters of interest and comparison to analytical predictions for the critical subsystems. Reference to who, where and when these tests and comparisons were performed.	Passed	The development of the Papyrus toolset has been issued in industrial and research collaborative projects, e.g. ATESST[5], INTESTRESTED[6]. Below are also some papers that reference the Papyrus core features: *F. Lagarde, H. Espinoza, F. Terrier, and S. Gérard. 2007. Improving uml profile design practices by leveraging conceptual domain models. ASE '07. ACM, pp. 445-448.* *M. Faugere, T. Bourbeau, R. d. Simone and S. Gerard, "MARTE: Also, an UML Profile for Modeling AADL Applications," ICECCS 2007, pp. 359-364.* *Gérard S., Dumoulin C., Tessier P., Selic B. (2010) 19 Papyrus: A UML2 Tool for Domain-Specific Language Modeling. MBEERTS 2007. LNCS, vol 6100. Springer* *S. Gérard, "Once upon a Time, There Was Papyrus…," MODELSWARD 2015, pp. IS-7.*
TRL 4: Functional Verification (Laboratory)	System concepts that have been considered and results from testing laboratory-scale breadboard(s). References to who did this work and when. Provide an estimate of how breadboard hardware and test results differ from the expected system goals.	Passed	Papyrus components are released as an Eclipse open-source project since decade. The verification of the integration has been managed by the Papyrus development and through feedbacks from the open-source community and users. *S. Demathieu, F. Thomas, C. André, S. Gérard and F. Terrier, "First Experiments Using the UML Profile for MARTE,"* ISORC *2008, pp. 50-57.* *Lagarde F., Espinoza H., Terrier F., André C., Gérard S. FASE 2008. LNCS, vol 4961. Springer*

[5] ATESST project: https://cordis.europa.eu/project/rcn/87278/factsheet/fr

[6] INTERESTED project: https://cordis.europa.eu/project/rcn/85281/factsheet/es

Level	Supporting information	Status	Justifications
TRL 5: Functional Verification (Representative)	Results from testing laboratory breadboard system are integrated with other supporting elements in a simulated operational environment. How does the "relevant environment" differ from expected operational environment? How do the test results compare with expectations? What Problems if any were encountered? Was the breadboard system refined to more nearly match the expected system goals?	Passed	The Papyrus features have been used to tackle several system engineering concerns, e.g. timing analysis, requirement engineering, code generation, etc. and endorsed to support several OMG specifications. Representative test cases have been performed by research and industrial project partners (e.g. ATESST, EDONA [55], MAENAD[7]) not involved in Papyrus development team models for the platform features verification/validation. Some results are available in papers, e.g.: *H. Dubois, F. Lakhal and S. Gérard, "The Papyrus Tool as an Eclipse UML2-modeling Environment for Requirements," RE@MARKE* 2009, *pp. 85-88.* *Ober, I., Baelen, S.V., Graf, S., Filali, M., Weigert, T., & Gérard, S. (2008). Model Based Architecting and Construction of Embedded Systems. MoDELS Workshops. Revol S. et al. (2008) Unifying HW Analysis and SoC Design Flows by Bridging Two Key Standards: UML and IP-XACT. DIPES 2008, vol 271. Springer* *Anssi S., Gérard S., Albinet A., Terrier F. (2011) Requirements and Solutions for Timing Analysis of Automotive Systems. SAM 2010. LNCS, vol 6598. Springer* *C. Mraidha, S. Tucci-Piergiovanni, and S. Gerard. 2011. Optimum: a MARTE-based methodology for schedulability analysis at early design stages. SIGSOFT Softw. Eng. Notes 36, 1 (January 2011)* *Dumitrescu C., Tessier P., Salinesi C., Gérard S., Dauron A., Flexible Product Line Derivation Applied to a Model Based Systems Engineering Process. CSDM 2013. Springer* *Cancila, D., Terrier, F., Belmonte, F., Dubois, H., Espinoza, H., Gérard, S., and Cuccuru, A. 2009. Sophia: A modeling language for model-based safety engineering. In Proceedings of the ACES-MB 2009, pp. 11--26.*

[7] MAENAD project: http://www.maenad.eu/

Level	Supporting information	Status	Justifications
TRL 6: Performance demonstration (Relevant or simulated)	Results from laboratory testing of a prototype system that is near the desired configuration in terms of performance, weight, and volume. How did the test environment differ from the operational environment? Who performed the tests? How did the test compare with expectations? What problems, if any, were encountered? What are/were the plans, options, or actions to resolve problems before moving to the next level?	Passed	Papyrus have been successfully used and they are still currently used by industrial partners (e.g. Airbus, Safran, Ericsson) in several relevant case studies of different research project (e.g. Robmosys[8], AQUAS[9], Vessedia[10]) and in different domains (e.g. automotive, railway, robotic, etc.). The Papyrus modelling features have also been extensively customized, e.g. in CHESS project, ESF (Eclipse Safety Framework) project[11], EAST-ADL2 related projects. Some results are available in papers, e.g.: *Lortal G., Dhouib S., Gérard S., Integrating Ontological Domain Knowledge into a Robotic DSL. MODELS 2010. LNCS, vol 6627. Springer* *Taha S., Radermacher A., Gérard S. An Entirely Model-Based Framework for Hardware Design and Simulation. DIPES 2010, BICC 2010, vol 329. Springer* *E. Wozniak, C. Mraidha, S. Gerard and F. Terrier, "A Guidance Framework for the Generation of Implementation Models in the Automotive Domain,"* SEAA *2011, pp. 468-476.* *D. Lugato, C. Bigot, Y. Valot, Validation and automatic test generation on UML models: the AGATHA approach, Electronic Notes in Theoretical Computer Science, Volume 66, Issue 2, 2002, Pages 33-49* *Cuenot P. et al. (2007) Towards Improving Dependability of Automotive Systems by Using the EAST-ADL Architecture Description Language. Architecting Dependable Systems IV. LNCS, vol 4615. Springer*

8 Robmosys project: https://robmosys.eu/

9 AQUAS project: https://aquas-project.eu/

10 Vessedia project: https://www.vessedia.eu/

11 Eclipse Safety Framework: https://www.polarsys.org/esf/

Level	Supporting information	Status	Justifications
TRL 7: Performance demonstration (Operational environment)	Results from testing a prototype system in an operational environment. Who performed the tests? How did the test compare with expectations? What problems if any, were encountered? What are/were the plans, options, or actions to resolve problems before moving to the next level?	Passed	Papyrus is operationally deployed in some companies, (e.g. Ericsson, Plastic omnium, Safran) [40][41][42]. Some testimonials are referenced on Papyrus website[12]. Papyrus has also been used as basic building blocks of some commercial modelling tools on the market, e.g. SCADE[13], Physistem[14], CIL4Sys simulator[15] .

Papyrus offers also an integration with the CDO repository. However, this feature has not been enough experienced on real use cases. This feature can be assessed at **TRL 3** as proof of concept.

5.2.2 Papyrus new feature

In the context of AMASS, some developments have been made to improve the existing architectural pattern support feature of Papyrus to cope with AMASS objectives/STOs. This feature had been developed in the context of SIRSEC project and led to a publication [56]. Table 9 presents a TRL evaluation for this feature that has been assessed at a level of maturity of **TRL 4**.

Table 9. TRL assessment of Papyrus feature developed in AMASS

Level	Supporting information	Status	Justifications
TRL 1: Basic Principles	Published research that identifies the principles that underlie this technology. References to who, where, when.	Passed	The principles of architectural pattern support have been first developed in the SIRSEC project[16].
TRL 2: Concept formulation	Publications or other references that outline the application being considered and that provide analysis to support the concept.	Passed	A conceptual approach was proposed in the following publication: *A. Radermacher, B. Hamid, M. Fredj, and J-L. Profizi. 2015. Process and tool support for design patterns with safety requirements. EuroPLoP '13.*

12 Papurus website: https://www.eclipse.org/papyrus/index.php#applications

13 Sace suite: http://www.esterel-technologies.com/products/scade-suite/

14 Phisystem tool: https://www.sherpa-eng.com/produits/phisystem/

15 CIL4Sys: http://cil4sys.com/simulator

16 SIRSEC project: https://systematic-paris-region.org/fr/projet/sirsec/

TRL 3: Proof of concept	Results of laboratory tests performed to measure parameters of interest and comparison to analytical predictions for the critical subsystems. Reference to who, where and when these tests and comparisons were performed.	Passed	The design pattern development, associated Papyrus customization, as well as a subset of GoF (Gang of Four) and safety patterns have been made by the Papyrus development team. An improvement has been performed during the AMASS project.
TRL 4: Functional Verification (Laboratory)	System concepts that have been considered and results from testing laboratory-scale breadboard(s). References to who did this work and when. Provide an estimate of how breadboard hardware and test results differ from the expected system goals.	Passed	In AMASS, the Papyrus feature has been integrated and customized with CHESS related development to define some AMASS architectural patterns related features (see D3.6 [36]).

5.3 TRL Assessment: CHESS Plugin for Papyrus

5.3.1 CHESS Plugin for Papyrus

The CHESS plugin for Papyrus augments Papyrus models with dependability information, initiates analyses of the model, and stores results from the analyses. CHESS core functionalities have been developed in the context of severañ research projects, basically CHESS[17], SafeCer [2] and CONCERTO[18]. The aforementioned functionalities are:

- CHESS modelling language support, for dependable, real time and contract-based properties modelling (as customization/profiling of the UML, SysML and MARTE standard languages)
- Support for schedulability analysis
- Support for dependability analysis: failure propagation and state-based analysis
- Support for contract refinement analysis
- Support for Ada code generation

The TRL for the CHESS core functionalities can be classified at a level of maturity of **TRL 6**; justifications are provided in the following Table 10.

17 CHESS project: http://www.chess-project.org/, 2014

18 CONCERTO project: www.concerto-project.org/ 2016

Table 10. Core Papyrus/CHESS plugin TRL assessment

Level	Supporting information	Status	Justifications
TRL 1: Basic Principles	Published research that identifies the principles that underlie this technology. References to who, where, when	Passed	The principles which are the basis of the CHESS approach (model-based design and performance, contract-base and dependability analysis) have been reported in several papers, e.g.: *A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio, A. Zovi and T. Vardanega, CHESS: A Model-Driven Engineering Tool Environment for Aiding the Development of Complex Industrial Systems, ASE 2012.* *L. Baracchi, A. Cimatti, G. Garcia, S. Mazzini, S. Puri and S. Tonetta, Requirements refinement and component reuse: the FoReVer contract-based approach, in A. Bagnato, I. R. Quadri, M. Rossi and l. S. Indrusiak, Editors Industry and Research Perspectives on Embedded System Design, IGI Global, Hershey PA, USA 2014.* *B. Gallina; E. Sefer; A. Refsdal: Towards Safety Risk Assessment of Socio-Technical Systems via Failure Logic Analysis, ISSRE Workshops 2014.*
TRL 2: Concept formulation	Publications or other references that outline the application being considered and that provide analysis to support the concept.	Passed	Practical applications of the CHESS approach have been presented in several research projects and papers, e.g.: *A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio, A. Zovi and T. Vardanega, CHESS: A Model-Driven Engineering Tool Environment for Aiding the Development of Complex Industrial Systems, ASE 2012.* *L. Baracchi, A. Cimatti, G. Garcia, S. Mazzini, S. Puri and S. Tonetta, Requirements refinement and component reuse: the FoReVer contract-based approach, in A. Bagnato, I. R. Quadri, M. Rossi and l. S. Indrusiak, Editors Industry and Research Perspectives on Embedded System Design, IGI Global, Hershey PA, USA 2014.* *Barbara Gallina; Edin Sefer; Atle Refsdal: Towards Safety Risk Assessment of Socio-Technical Systems via Failure Logic Analysis, ISSRE Workshops 2014.*
TRL 3: Proof of concept	Results of laboratory tests performed to measure parameters of interest and comparison to analytical predictions for the critical subsystems. Reference to who, where and when these tests and comparisons were performed.	Passed	The development of the CHESS toolset core components has been performed mainly during the CHESS and CONCERTO research projects.

Level	Supporting information	Status	Justifications
TRL 4: Functional Verification (Laboratory)	System concepts that have been considered and results from testing laboratory-scale breadboard(s). References to who did this work and when. Provide an estimate of how breadboard hardware and test results differ from the expected system goals.	Passed	In CONCERTO and SafeCer new components have been integrated with the CHESS core ones, in particular regarding improvement of schedulability analysis (support for multi core) and support for contract-based design. The verification of the integration has been managed by the CHESS development team.
TRL 5: Functional Verification (Representative)	Results from testing laboratory breadboard system are integrated with other supporting elements in a simulated operational environment. How does the "relevant environment" differ from expected operational environment? How do the test results compare with expectations? What Problems if any were encountered? Was the breadboard system refined to more nearly match the expected system goals?	Passed	The core CHESS features have been tested during the CHESS, CONCERTO and SafeCer projects. Representative models have been used as input to the defined test cases. The verification of the test cases has been performed by partners not involved in CHESS.
TRL 6: Performance demonstration (Relevant or simulated)	Results from laboratory testing of a prototype system that is near the desired configuration in terms of performance, weight, and volume. How did the test environment differ from the operational environment? Who performed the tests? How did the test compare with expectations? What problems, if any, were encountered? What are/were the plans, options, or actions to resolve problems before moving to the next level?	Working	Papyrus (for what regards the modelling support) and CHESS (as additional modelling and analysis capabilities on top of Papyrus) have been successfully used and they are still currently used by industrial partners in several relevant case studies of different research project (e.g. CONCERTO, SafeCOP[19], AQUAS[20]), both as modelling and analysis environment. Some results about CHESS are available in papers, e.g.: *Model-based design and schedulability analysis for avionic applications on multicore platforms: www.cister.isep.ipp.pt/docs/CISTER-TR-160610, 2016* *S. Mazzini; S. Puri; A. Russino: fitting the CHESS Approach to the AUTOSAR Development Flow. Source: Ada User Journal. Sep2016, Vol. 37 Issue 3, p150-156. 7p.*

[19] SafeCOP project: http://www.safecop.eu/
[20] AQUAS project: https://aquas-project.eu/

5.3.2 CHESS new features

The CHESS extensions made in the context of the AMASS project can be summarized as:

- Support for modelling:
  - Additional views to support contract-based modelling and contract properties specification
  - Argumentation generation from CHESS model
  - Parameterized architecture
  - Architectural patterns
- Analysis:
  - Improvement of model-based Fault tree analysis
  - Contract-based analysis
  - Contract-based fault tree and FMEA analysis
  - Trade-off analysis
  - V&V Manager
  - Store results from the analysis
- Support for assurance case:
  - Document generation
  - Argument fragment generation
  - Traceability links

Table 11 below presents an evaluation of the TRL of the aforementioned CHESS extensions for Papyrus, that have achieved a **TRL 5** level of maturity.

Table 11. AMASS Papyrus/CHESS plugin TRL assessment

Level	Supporting information	Status	Justifications
TRL 1: Basic Principles	Published research that identifies the principles that underlie this technology. References to who, where, when.	Passed	Ruiz, A., Gallina, B., de la Vara, J.L., Mazzini, S., Espinoza, H.: Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems. SASSUR 2016.
TRL 2: Concept formulation	Publications or other references that outline the application being considered and that provide analysis to support the concept.	Passed	The conceptual part is included in the CACM and in D3.3 [43]. Concerning the considered applications, AMASS D3.8 [51]and the following publications are also relevant: *Sljivo, I., Juez, G., Puri, S., Gallina, B.: Guiding Assurance of Architectural Design Patterns for Critical Applications. Ada User Journal (accepted paper)* *Alaña, E., Herrero. J.: Design and Safety assessment of on-board software applications using the AMASS platform. EUROSPACE - DASIA 2018*

Level	Supporting information	Status	Justifications
TRL 3: Proof of concept	Results of laboratory tests performed to measure parameters of interest and comparison to analytical predictions for the critical subsystems. Reference to who, where and when these tests and comparisons were performed.	Passed	The development of the CHESS toolset extensions has been performed during the AMASS research project.
TRL 4: Functional Verification (Laboratory)	System concepts that have been considered and results from testing laboratory-scale breadboard(s). References to who did this work and when. Provide an estimate of how breadboard hardware and test results differ from the expected system goals.	Passed	In AMASS new components have been integrated with the CHESS core ones, e.g. regarding basic functionality (ad-hoc views, creation of diagrams) and regarding new analysis support (e.g. model-based safety analysis). Information about the architecture has been provided in AMASS D3.6 [36]. The verification of the integration has been managed by the CHESS development team.
TRL 5: Functional Verification (Representative)	Results from testing laboratory breadboard system are integrated with other supporting elements in a simulated operational environment. How does the "relevant environment" differ from expected operational environment? How do the test results compare with expectations? What Problems if any were encountered? Was the breadboard system refined to more nearly match the expected system goals?	Passed	The core CHESS features have been tested during the AMASS core platform validation, as documented in D2.6 [38]. The final version of the new CHESS features/components implemented in AMASS has been tested during the AMASS P2 platform validation, as documented in D2.8 [39]. Representative models have been used as input to the defined test cases. The verification of the test cases has been performed by partners not involved in CHESS. The new features implemented in AMASS have been evaluated in the context of different AMASS case studies implementation; bugs have been found and fixed.

Another CHESS extension made in AMASS related to the "Support for modelling" category is the following:

- Collaborative modelling using CDO

The aforementioned extension has been made available by exploiting the Papyrus support for CDO, and by allowing the export of CHESS projects from file-based to CDO (and vice versa). Regarding working with Papyrus/CHESS models in CDO, some problems have been raised by AMASS case studies implementers; in particular, some diagrams available in the file-based project have become corrupted when migrated to CDO. It has not been possible to provide a full patch for the aforementioned issue, mainly due to the fact that the issue seems also related to the integration of the CDO and UML libraries available in Eclipse Neon (the Eclipse distribution used in AMASS). This issue will be investigated in the context of one of the latest Eclipse/Papyrus/CDO release and, if replicable in such environment, a bug will be reported to the Papyrus and CDO Eclipse projects. So, regarding this aspect, the TRL for the "Collaborative modelling using CDO" feature is evaluated to **TRL 3**.

5.4 TRL Assessment: OpenCert

5.4.1 OpenCert tool

The OpenCert core functionalities have been developed in the context of the OPENCOSS[21] research project. The aforementioned functionalities are:

- Standards & Regulations Information Management
- Assurance Project Management
- Assurance Case Management
- Evidence Management
- Compliance Management
- Cross-Domain and cross project reuse
- Web reports
- CDO repository for model's storage

The TRL assessment of the OpenCert tool prior to AMASS was **TRL 4**. Some publications to justify this level are:

- A. Ruiz, A. Melzi, T. Kelly: Systematic Application of ISO 26262 on a SEooC, Design, Automation and Test in Europe (DATE), Grenoble, March 2015.
- Nair, S., de la Vara, J.L., Sabetzadeh, M., Falessi, D, Evidence Management for Compliance of Critical Systems with Safety Standards: A Survey on the State of Practice, Information and Software Technology 60: 1-15, Elsevier, 2015
- K. Attwood and T. Kelly, Controlled Expression for Assurance Case Development, Engineering Systems for Safety: Proceedings of the 23rd Safety-Critical Systems Symposium, Bristol, 2015.

5.4.2 OpenCert new features

The main OpenCert extensions made in the context of the AMASS project are:

- Migration of the core features to Eclipse Neon 3
- Adoption of GSN notation for Assurance Cases
- Reuse Assistant view for cross-standard and cross-project reuse
- OpenCert Perspective
- Access Management feature
- Improvements in Assurance Project generation process
- Improvements in compliance management
- Import of models from file into Assurance Projects in CDO
- Export of Assurance cases from CDO to file
- OpenCert Dashboard
- Integration with third-party tools (e.g. VERIFICATION Studio, OSLC-KM, OCRA, Safety Architect, Cyber Architect, SVN)

Table 12 presents an evaluation of the TRL of OpenCert new features developed in the context of AMASS project with the required justifications, that has resulted in a level of maturity of **TRL 5**.

[21] http://www.opencoss-project.eu/

Table 12. OpenCert TRL assessment

Level	Supporting information	Status	Justifications
TRL 1: Basic Principles	Published research that identifies the principles that underlie this technology. References to who, where, when	Passed	OpenCert has already functionalities background. As example: *Huáscar Espinoza, Alejandra Ruiz, Mehrdad Sabetzadeh, Paolo Panaroni: Challenges for an Open and Evolutionary Approach to Safety Assurance and Certification of Safety-Critical Systems, WOSOCER, Hiroshima, Nov 2011.* *Xabier Larrucea, Paolo Panaroni: A harmonized multimodel framework for safety environments, EuroSPI Conference 2012, Vienna.* *Ruiz, A., Gallina, B., de la Vara, J.L., Mazzini, S., Espinoza, H.: Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems. SASSUR 2016.*
TRL 2: Concept formulation	Publications or other references that outline the application being considered and that provide analysis to support the concept.	Passed	The conceptual part is included in the CACM (Appendix B of this document) and in the following documents D2.4 [7], D3.3 [43], D4.3 [44], D5.3 [45] and D6.3 [46]. Also, the following publication is relevant: *Espinoza, H., de la Vara, J.L., Juez, G., Martinez, C., Gallina, B., Puri, S., Mazzini, S., Blondelle, G.: Meet the new Eclipse-based tools for Assurance and Certification of Cyber-Physical Systems. Eclipse Newsletter July 2018*
TRL 3: Proof of concept	Results of laboratory tests performed to measure parameters of interest and comparison to analytical predictions for the critical subsystems. Reference to who, where and when these tests and comparisons were performed.	Passed	The developers have done specific validation and training with videos explaining[22] and demonstrating[23] the functionalities.

22 https://www.amass-ecsel.eu/content/training

23 https://www.amass-ecsel.eu/content/demos

TRL 4: Functional Verification (Laboratory)	System concepts that have been considered and results from testing laboratory-scale breadboard(s). References to who did this work and when. Provide an estimate of how breadboard hardware and test results differ from the expected system goals.	Passed	Functional testing has been done and publish in AMASS deliverable D2.6 [38], D2.7 [20] and D2.8 [39]. Different testers have participated in the different test campaigns. Also, some publications were released: *Nair, S., de la Vara, J.L., Sabetzadeh, M., Falessi, D: Evidence Management for Compliance of Critical Systems with Safety Standards: A Survey on the State of Practice, Information and Software Technology 60: 1-15 (2015).*
TRL 5: Functional Verification (Representative)	Results from testing laboratory breadboard system are integrated with other supporting elements in a simulated operational environment. How does the "relevant environment" differ from expected operational environment? How do the tests result compare with expectations? What Problems if any were encountered? Was the breadboard system refined to more nearly match the expected system goals?	Passed	The code has been published in a public Git repository[24]. OpenCert has been used in AMASS case studies, more specifically in CS1, CS3, CS4, CS7, CS8, CS9, CS10, CS11 [49]. The results of the use of the tool on the case studies have been published in deliverables D1.4 [47], D1.5 [48] and expected to be also included in D1.6. Also, some publications were released: *Alejandra Ruiz, Alberto Melzi, Tim Kelly: Systematic Application of ISO 26262 on a SEooC, DATE 2015, Grenoble, March 2015.*
TRL 6: Performance demonstration (Relevant or simulated)	Results from laboratory testing of a prototype system that is near the desired configuration in terms of performance, weight, and volume. How did the test environment differ from the operational environment? Who performed the tests? How did the test compare with expectations? What problems, if any, were encountered? What are/were the plans, options, or actions to resolve problems before moving to the next level?	In progress	The results of the use of the tools in the case studies is still in progress.

24 git.polarsys.org/c/opencert/opencert.git

5.5 TRL Assessment: Integration of EPF Composer and BVR Tool with other AMASS Tools

5.5.1 EPF Composer and BVR tool

EPF Composer is the only available implementation of OMG's SPEM 2.0, but the migration of EPF Composer to newer versions of technologies was never performed. Accordingly, we evolved the EPF Composer from Eclipse Galileo 3.5.2 to Eclipse Neon 4.6.3 after 11 years [57]. This was done for performing the integration with other tools in the AMASS platform. For a complete list of all enhancements and defects addressed in the EPF Version 1.5.2 [59], please see the Bugzilla page [60]. It might be noted that the migration has been tested by the EPF Composer team of IBM. Likely, the EPF Composer qualifies for **TRL 7**.

The BVR tool is developed in the context of the EU VARIES (VARiability In safety-critical Embedded Systems) project [58]. The BVR Tool qualifies for the **TRL 5**.

5.5.2 EPF Composer new features

In the context of the AMASS project, the migration EPF Composer from Eclipse Galileo 3.5.2 to Eclipse Neon 4.6.3 was performed in four steps:

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

In the context of the AMASS project, some plugins have been implemented for supporting the communication between EPF Composer, OpenCert and BVR Tool. They are mentioned below:

- The **Fallacy Detection plugin** takes as input the process and standard requirements and validates whether the process contains the sufficient information corresponding to the key evidence for supporting the specific requirements.
- The **Process-based Argument Generator plugin** takes the modified process as an input and transforms it into arguments (model and diagram) in OpenCert.
- The plugin for the generation of **Executed Process Models** takes a delivery process modelled in EPF Composer and generates an evidence model and a process model in OpenCert
- The **Requirement Transformation plugin** transforms the standard's requirement modelled in EPF Composer to the Baseline (model and diagram) in OpenCert.
- The **Process Lines plugin** provides support for importing backend folders and files within the method library of EPF Composer, resolving problems with the files for variability management with the BVR Tool, and exporting back the resolved process models to the EPF Composer.

Table 13 presents an evaluation of the TRL for the new features developed for the integration of EPF Composer and BVR in the context of the AMASS project with the required justifications, that has resulted in a level of maturity of **TRL 5**.

Table 13. EPF+BVR TRL assessment

Level	Supporting information	Status	Justifications
TRL 1: Basic Principles	Published research that identifies the principles that underlie this technology. References to who, where, when	Passed	The development processes modelled in EPF Composer needs to fulfil the standards and should provide the justification of compliance. The arguments are derived directly from the Process models. To prevent a fallacious derivation of arguments, a common type of fallacy (Key-evidence omission) is detected before enabling the generation. For facilitating the compliance between the executed process (including the corresponding evidence) and the planned process, the transformations of standards requirements and planned process from EPF Composer into baselines, post-planning process and evidence models into OpenCert are performed. Process variability is modelled and realized with the BVR Tool. In this regard, the seamless integration bridges the gap by resolving problems in EPF Composer models and export back of configured process models into the EPF Composer.
TRL 2: Concept formulation	Publications or other references that outline the application being considered and that provide analysis to support the concept.	Passed	Four papers have been published for the dissemination of achieved results. The best paper in QUATIC 2018 is titled as "*Preventing Omission of Key Evidence Fallacy in Process-based Argumentations"* in 11th International Conference on the Quality of Information and Communications Technology (QUATIC '18), Coimbra, Portugal, September 4-7, 2018, authors: Faiz Ul Muram, Barbara Gallina and Laura Gómez Rodríguez. The paper in RSSRail 2019 is titled as "*A Tool-supported Model-based Method for Facilitating the EN50129-compliant Safety Approval Process"* in 3rd International Conference Reliability, Safety and Security of Railway Systems: Modelling, Analysis, Verification and Certification (RSSRail '19), Lille, France, June 4-6, 2019, authors: Faiz Ul Muram, Barbara Gallina and Samina Kanwal. The tool paper in SPLC 2018 is titled as *"Safety-oriented process line engineering via seamless integration between EPF Composer and BVR Tool"* in 22nd International Systems and Software Product Line Conference (SPLC '18), Gothenburg, Sweden, September 10-14, 2018, authors: M. A. J. and Barbara Gallina. The paper in SAC 2019 is titled as "*Towards Variant Management and Change Impact Analysis in Safety-oriented Process-Product Lines"* in 34th Annual ACM Symposium on Applied Computing (SAC '19), Limassol, Cyprus, April 8-12, 2019, authors: M. A. Javed, B. Gallina and A. Carlsson.

TRL 3: Proof of concept	Results of laboratory tests performed to measure parameters of interest and comparison to analytical predictions for the critical subsystems. Reference to who, where and when these tests and comparisons were performed.	Passed	The applicability of the integration been EPF Composer and BVR Tool has been demonstrated in the Tool Demonstration session of SPLC 2018, Gothenburg, Sweden, September 10-14, 201. The basic concepts have also been highlighted and presented in a poster at EclipseCon 2018: *M. A. Javed and B. Gallina. Get EPF Composer back to the future: A trip from Galileo to Photon after 11 years. EclipseCon, Toulouse, France, June 13-14, 2018*. Besides the 4 pages in SAC 2019 proceedings, a poster will be presented for the demonstration of basic principles and concepts. Nevertheless, the poster has also been presented to the interested participants of SafeComp 2018.
TRL 4: Functional Verification (Laboratory)	System concepts that have been considered and results from testing laboratory-scale breadboard(s). References to who did this work and when. Provide an estimate of how breadboard hardware and test results differ from the expected system goals.	Passed	The functional verification has been performed at the AMASS P2 bundle. EPF Composer and BVR Tool are open source and well-known process engineering and variability management solutions based on OMG's SPEM and CVL standards, respectively. The former is an open source version of IBMs project (called Rational Method Composer). It is a mature tool, which is widely adapted worldwide. The BVR Tool was developed in the context of an EU project (called VARIES[25]). Functional testing has been done and publish in AMASS deliverable D2.8 [39]. Different testers have participated in the different functionalities.
TRL 5: Functional Verification (Representative)	Results from testing laboratory breadboard system are integrated with other supporting elements in a simulated operational environment. How does the "relevant environment" differ from expected operational environment? How do the test results compare with expectations? What Problems if any were encountered? Was the breadboard system refined to more nearly match the expected system goals?	Passed	The evaluation of the fallacy detection and transformation from EPF Composer to OpenCert is performed and demonstrated for the industrial cases from OHB and Alstom in AMASS. The evaluation of the integration between EPF Composer and BVR Tool is performed and demonstrated for the industrial cases from OHB, VIF, Infineon and LAN in AMASS[26]. Besides that, the trainings have been given to the OHB, VIF and LAN for the usage (from 2017 to 2019). The test results fulfilled the expectations.

25 https://artemis-ia.eu/project/42-varies.html

26 Demo video: https://www.youtube.com/watch?v=_bONHtkRPEA

6. AMASS Public Artefacts Assessment

This section presents an assessment of the AMASS "public artefacts" from an external user point of view, i.e. a user that has not been involved in the AMASS project. The term "public artefact" makes reference to the AMASS artefacts presented in the AMASS Platform User Manual [5] that are summarized in Table 14.

Table 14. Sections of the AMASS User Manual with the analysed public artefacts

Related section of the User Manual [5]	Artefact
3	Dashboard Overview
4	Process Modelling with EPF Composer
5	Standards Modelling
6	Assurance Project Management
7	System Component Specification
8	System Dependability Co-analysis
9	Assurance Argumentation Management
10	Evidence Management
11	Functionalities of the Polarsys OpenCert Platform Server
12	Engineering of Process, product and Assurance Case Lines

First, we present the scope of the analysis. Then, we provide an overview of these public artefacts, as per the order of the sections of the User Manual [5]. Finally, we conclude with some feedback about the artefacts obtained from the analysis.

6.1 Scope of the Analysis

We describe the scope of the analysis by first asking the following questions:

- What is usability in this context?
- What is the intended user group?
- What are the limitations of this assessment?

Then, we select a case study on which we base the assessment.

6.1.1 Usability of the AMASS User Guidance and Methodological Framework

We evaluate the usability with respect to D2.5 [4] and the AMASS websites[27]. Usability can be defined as the "*extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*" (ISO 9241-11, 2018[28]). In general, methods for assessing usability are categorized as analytical or empirical. Analytical (or inspection) methods are used for interface inspection and perceived as a quick and low-cost solution. Empirical methods, on the other hand, test with actual users. "*Heuristic evaluation (HE)* – an informal, cheap and quick method where a small group of usability evaluators inspect a user interface to find and rate the severity of usability problems using a set of usability principles or heuristics. It enables the identification of major and minor problems and can be used early in the development process. Its disadvantages are that evaluators have to be experts to provide good results and identification of domain-specific problems is not reliable."[50].

[27] https://polarsys.org/opencert and https://www.amass-ecsel.eu

[28] https://www.iso.org/standard/63500.html

The AMASS artefacts analysis falls under the category of heuristic evaluation. Its goal is to assess the usability from the perspective of any new stakeholder who has not been involved in the project yet. The analysis has been specifically performed by an outsider having not been involved with the AMASS project and never been using the AMASS Platform before, yet being acquainted with safety and security engineering in other projects (both academic and industrial) along with various tools.

6.1.2 Target User Group of the AMASS Platform and this assessment

The AMASS Platform is intended for use (amongst others) by safety and security engineers, asserting features on large scale projects. As the D2.5 deliverable [4] points out on page 8, the AMASS Platform targets projects on the scale of the passenger plane 787, for which certification lasted eight years and consumed 200,000 person hours for technical work, targeting more than 1,500 regulations containing more than 4,000 documents for evidence. The AMASS websites (cited above) state “automotive, railway, aerospace, space, and energy” as example target industries.

The assessment has been conducted from the perspective of an engineer who is trained in other tools than AMASS open tool platform and who must start using the tool right away in a hurry with little time for training and with no support from other experts, developers or previous users. The assessment is supposed to be critical and independent, to point out potential for future improvements.

6.1.3 Limitations

This analysis was assigned to 44 person hours, which includes consulting the relevant AMASS deliverables[29], as well as the websites and related literature, i.e. the publications associated with the AMASS project. The reason to limit the time is that assessing suitable platforms commonly suffers from similar limitations. Companies carrying out large projects have large amount of human resources. Having them idling while the management selects a tool is therefore limited. The person did the assessment with an open mind as he had no previous relation with the project, not involved in conceptual discussions and had no relation during the assessment with previous users or developers or the features.

Possibilities for assessing the platform by conducting larger case studies with multiple engineers are here inherently confined. Also, comparison with competitor platforms is not possible. The focus is on assessing the usability in terms of starting to conduct a small assurance project focusing on one functionality, following an example of process development as described in Sections 2-12 of the AMASS Platform User Manual [5].

6.1.4 Case Study

The usability of AMASS shall be assessed by conducting a confined case study. To optimize the efforts, the study is selected from the case studies included at that time in the common AMASS database used for training, validation and case studies development. The AMASS deliverables offer all relevant documents. The number of documents yet makes it hard to find appropriate functionality candidates. Deliverables D3.8 [51], D4.8 [52] and D6.8 [54] contain appropriate case studies. Some of the functionalities for which the assurance workflows are discussed (cf. D4.8 Chapter 4) are:

- CS11 (Orbit Control System, Section 4.1) and
- CS3 (Cooperative Adaptive Cruise Control CACC, Section 4.2).

Both are also available in the AMASS SVN repository. We select the case study of the CS3 Cooperative Adaptive Cruise Control [49] for the assessment.

[29] https://www.amass-ecsel.eu/content/deliverables

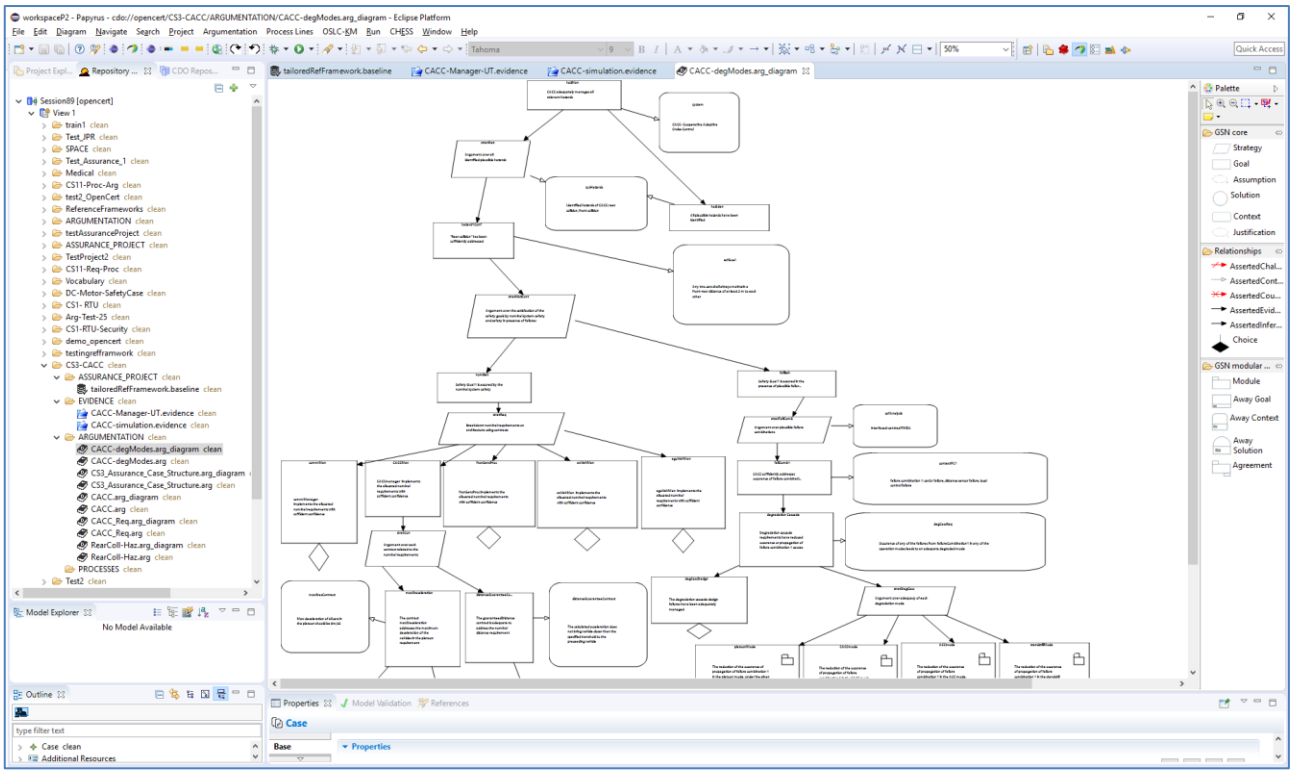


Figure 9. CS3 - CACC Case Study

6.1.5 Setup

The website[30] as well as the User Manual [5] (in its Section 2) both offer a setup guide that is easy to find and easy to follow. The setup of the AMASS Platform client bundle comprises three steps: download[31], decompress and setup the connection to the server. The package is 1.04GB in size and contains a customized Eclipse IDE (Version: Neon.3 (4.6.3), Build id: M20170301-0400, P2 Prototype) containing any library required.

The AMASS website stops being helpful after the setup. A link to the two documents used in this assessment for continuation is provided:

- The D2.5 AMASS User guidance and Methodological framework [4]
- The AMASS Platform User Manual (same file, starting at page 89).

The latter continues where the guide on the website ends with explaining the dashboard (Section 3).

6.2 Artefacts Overview

6.2.1 Dashboard

After the easy setup, the Section 3 of the AMASS Platform User Manual [5] introduces the Dashboard overview. At the time of this assessment (January 2019), the manual mentions the Dashboard Overview being still under development (November 2018), so it was not yet available in the AMASS Prototype P2. Hence, this section is limited to assessing the design.

The Dashboard is designed to provide a way to facilitate the user interaction with the platform. The manual

30 https://www.polarsys.org/opencert/resources/gettingstarted/, last accessed 14 January 2019

31 https://polarsys.org/opencert/downloads/

shows a glimpse of the look and feel of the Dashboard. It provides an overview over the assurance process, tools and capabilities, taking the user by the hand as a guide. It structures the workflow into Sections that are identical to the arrow-shaped boxes advertised throughout AMASS, from Standards Compliance Definition to Evidence Management.

The idea of a guided mode is very helpful, especially since large projects bring together stakeholders from many domains, none of which can be expected to have a complete overview.

6.2.2 Process Modelling with EPF Composer

The Section 4 of the User Manual [5] provides a brief overview of the EPF Composer tool. Untrained engineers have to consult the original manual which contains a thorough introduction with hours' worth of tutorials, which is also referenced in Section 4.

6.2.3 Standards Modelling

A de-facto start of the tool chain is the modelling of standards. One of the major benefits of the standards modelling artefact is the possibility of reusing assets between standards thanks to the equivalence mapping and Cross-standards reuse feature.

The corresponding feature is available in the AMASS Platform as "Reframework". It allows to formulate artefacts in natural language and define categories (e.g. criticality) as needed. The mapping tool then allows to map artefacts, basically linking them with annotations (i.e. filters).

Despite the mapping tool, the possibility for creating applicability tables (Section 5.4 of the User Manual [5]) is important. The result is one table with two sections, one for activity applicability and one for requirement applicability. The guide is very thorough with 30 figures and illustrations on 17 pages.

6.2.4 Assurance Project Management

The Assurance Project Management is explained in the Section 6 of the AMASS Platform User Manual [5], on 47 pages with 77 figures. It is, like the Reframework, also provided as a feature within the AMASS Platform. When browsing the folders that come with the installation, the following folders stick out:

- Argumentation
- Assurance Project
- Evidence
- Process

These folders are created when an assurance project is created. The assurance project has the task to follow the assurance process through the whole lifecycle.

It becomes obvious that using the tool for the first time can be tedious. Each desired action has first to be located and then to be created. The guide shows very detailed the single steps. Yet the sheer number of steps required to create standards, equivalence maps and so forth (earlier sections) and now a complete responsibility mapping is huge. Those tasks are done only once, when a new standard should be considered in the organization, so it not expected to be a daily task. On the other hand, it can be expected that once one is trained in using the tool, initial creation will be a lot faster. The dashboard can also help fast the learning curve.

6.2.5 System Component Specification

One foundation of the AMASS Platform can be seen in the tools CHESS and Papyrus, which help to model the system architecture. The focus of these tools lies in contract-based design (which aligns with assurance and standards compliance throughout the lifecycle discussed in the previous section). Both tools are well known in both academia and industry. One of the hinges complementing the AMASS project is reuse of

standards compliance via contracts. This means, that standards can be utilized to distil contracts for (automatized) reusable contract-based design.

The section 7 of the AMASS Platform User Manual [5] explains, on 44 pages with 59 figures, in detail the process: How CHESS comprises all relevant information about Papyrus, SysML, MARTE and about the available set of views, and how the tool can be used to build a model. The focus of the tool is on functional (i.e. qualitative) behaviour. Although it is possible to include probabilistic behaviour (e.g. Section 7.12.1, Probabilistic Fault Injection), a probabilistic analysis in the formal sense seems not possible (i.e. if fault injection is regarded as simulation). The strong suite is the functional analysis, reflected in fault tree analyses.

As valuable asset, CHESS allows to include consultation of external tools like OCRA. The detail in which Section 7 is presented is adequate, yet the sheer amount presented is very high. For instance, the Section 7.11 on Pattern design shows a structured path on page 121 that is easy to follow. Yet, to comprehend just Figure 161 showing an example, takes a long time. It contains a complex and real word example rather than a simple one. This is one feature which makes for a bit less than 1% of the whole manual. In this case, less would have been more. For some readers, start with a small consistent *hello-world* example would increase the read flow and decrease the loss of motivation to continue. However, for experiences users, this kind of complex example is highly appreciated.

6.2.6 System Dependability Co-analysis

On just three pages, the Section 8 of the AMASS Platform User Manual [5], discusses the co-analysis of multiple dependability criteria. What strikes the reader as one of the most interesting topics is the co-analysis of safety and security. However, the user manual includes information from the tool user perspective and not the conceptual work behind. The tool of choice in AMASS is CHESS. The bottom line is that security assurances are modelled like safety assurances and both can be analysed *in parallel*. An important pointer here is to D2.5 AMASS User guidance and Methodological framework [4], page 49. The reason why this is interesting is that both safety and security can have mutual implication. For instance, if a security layer is compromised, a different level of safety must be applied (e.g. another contract holds) and vice versa.

In another hand, one of the challenges on which companies like Volkswagen work is to harmonize safety and security to be able to assess both at the same time in the sense that mutual influence of hard-to-harmonize measures is achieved. AMASS can also help accomplish such a co-evaluation, while (functional) safety and security are assessed next to each other, so it would not make a difference if first safety and then security would be evaluated or the other way around.

6.2.7 Assurance Argumentation Management

One of the two final steps is the Assurance Argumentation Management. Once the models are created and tested, verified and validated, all the generated data must be organized.

On 37 pages, the Section 9 of the AMASS Platform User Manual [5], explains the tool that comprises mostly diagram drawing and editing with 41 figures. When learning the diagrams, page 157/158 are the ones to bookmark as they show an overview of the graphical notation. The next important part is the definition of vocabularies (Section 9.4) since standards and contracts often contain natural language elements. Formalizing them in a vocabulary can be quite an asset. The section closes by explaining the argumentation generation, leading to the final step.

6.2.8 Evidence Management

Once the arguments are generated, the collected evidence needs to be managed. The Section 10 of the AMASS Platform User Manual [5], like the ones before, is an excellent step-by-step guide, featuring 73

figures on 37 pages. At times it is too detailed, for instance on page 209 when it is explained that the Delete button deletes a property value, the real interesting part starts at Section 10.8, the Impact analysis.

6.2.9 Functionalities of the Polarsys OpenCert Platform Server

After the workflow is finished, big projects will have produced an enormous amount of data. Even the comparably small case studies already generate an impressive amount of data and huge models. All this data eventually must be distilled into a report. This is where the Web application hosted in the common Server introduced in Section 11 of the AMASS Platform User Manual [5], comes in. The server provides also a web server that has a web-site to show some data while the Desktop client can do “CRUD” operation over all the data. The steps are well explained.

6.2.10 Engineering of Process, product and Assurance Case Lines

The Section 12 of the AMASS Platform User Manual [5] continues where Section 4 (EPF Composer) left off. It targets the engineering of process, product and assurance case lines. The contribution of this chapter exceeds the scope of this assessment and is thereby omitted.

6.3 Feedback from the AMASS Public Artefacts assessment

In its current form, the AMASS Platform User Manual seems - like the tool itself - in a prototypical state. For example, section 3 requires attention, for the Dashboard not yet being implemented. This will change in the future when the Dashboard will be available. The whole flow can be improved with one *hello-world* example that is subsequently built throughout the sections and that is supplied for download as reference.

One general drawback of the platform is its responsiveness. While this might be due to the tool working with CDO, slow responsiveness coined the user experience during this assessment as well.

The common database seems overloaded for users on beginner level. Either supplying a minimalistic version or a beginner’s view with restricted capabilities (e.g. for just developing the *hello-world* example) would help. It is used for different type of users, for testing, validation, training and project case studies development so inconsistencies and incompleteness are usual.

One positive aspect is the library of examples synchronized from an online repository when setting it up. The CS3 - CACC example shows the potential of the platform and would not have been possible to be created within this assessment.

A further analysis that would be interesting to conduct is between AMASS and similar (and even less similar) platforms. It would be interesting to develop the same model assessment for different platforms to point out the limitations and strengths of each.

One improvement that could be adapted from (for instance) CounterExample-Guided-Abstraction-Refinement (as amongst others pioneered by Edmund Clarke) is the autonomous refinement process (with regards to Section 7.18 in AMASS Platform user Manual [5]). This means, once a contract failed in one context (e.g. project or w.r.t. a specific standard) but was fixed with some adaptation, the same adaptation could be carried out in an equivalent case for another context. When machine learning is applied, the system might even learn how to adapt contracts/requirements/system specifications on its own to succeed (in the far future).

Another promising domain is the integration of tools that do not target functional properties (and its limitations). How would for instance *measurable* properties like a probability distribution over temperatures be mapped to binary contracts that hold or do not hold? How would multi-critical contracts on continuous domains be modelled (e.g. Brownian motion) and their trade-offs (e.g. in terms of Pareto optimality) derived (targeting Section 3.6.6 in the D2.5 AMASS User guidance and Methodological framework [4])?

For dissemination means, the YouTube channel falls too short as it covers some features only on their surface, further videos are planned but were not public available at the time of the assessment. A dedicated video tutorial including tasks would improve the getting-to-know the system tremendously. The tutorial could be in the form of an online lecture that would also be targeting industrial partners. In addition, it could be interesting to define pagers that describe each feature of the AMASS Platform to let the users have a better understanding of the whole platform functionalities. The information could be made available to the AMASS community through its website, the User Manual, etc. An example of template for such a pager is presented in Table 15.

Table 15. Example of template pager for AMASS main features

Feature Name: ……
Intent: ……
Tool(s) support: ……
Estimated TRL: ……
Current strengths: ……
Current limitations: ……
Applicability (Necessary Context): ……
Example applications (where to find them): ……
Known uses: ……
Related features within the platform: ……
Seamless interoperability status: ……
Documentation:
Detailed guidance: ……
User manual section(s): ……
Scientific paper(s): ……
Training: ……
Contact person/institution: ……

7. Recommendations for Platform Usage and Evolution

Following the outcomes of the validation campaigns and findings from Tool qualification, TRL assessment and Public Artefacts assessment, we would make some recommendations for the usage and the evolution of the Platform. The recommendations concern mainly the usability of the platform and the support of the external tools.

- **About Usability**

  At the moment, there are safety related features at different places and some menus are even duplicated within the same place. The UI interface may be more user-friendly by avoiding a same feature with numerous menus at different places. It will be also worth to have a better organization of the menus, either per concern or per tool to facilitate the navigability of the user.

  There are several features embedded in the AMASS Platform. An integrated user guide from the Help menu of a dashboard will be useful to guide the user when using the tool. In addition, it will be great to provide within the platform full examples of projects demonstrating some features or tool chain usage.

  Another thread is the reduction of the AMASS Platform. The platform is very potent. It contains many tools that are nicely organized in a workflow like an assembly line, ready to serve both industrial and academic contexts. The dashboard will feature the assembly line, being very accessible. Yet, although the manual is easy to digest, it is too long. A minimal version of both the AMASS Platform and the manual for beginners with a shortened basic guide would be very helpful. Alternative to a reduced version of AMASS, a version blending out non-basic features like a *beginner's perspective* would improve accessibility like in the scope of this assessment. The AMASS public artefacts assessment was conducted in 44 hours (about one person week). Assessing all relevant literature and understanding the platform with all features is not feasible in that amount of time. The assessment was conducted by a person with no previous relation with the project, no knowledge of the conceptual discussion, and with no access to previous users or experts. The conclusions of the usability analysis performed by the CS owners are in line with the AMASS public artefacts assessment.

- **About Performance**

  The AMASS Platform is very slow, with some unresponsive features, especially when the tool is connected with a distant server behind a proxy and using models database storage technology such as CDO. However, it is expected that the platform performance will be improved while dedicated servers are in use within the company that deploys the tool. In fact, many case studies have opted for this choice and have a dedicated server for the case study development.

- **About Security**

  The AMASS platform provides collaborative features and storage through CDO, so some security questions can arise about the tool using the cloud technology. Security and performance could be improved using the tool through a VPN. However, as the AMASS Platform is intended to be deployed in a company in order to be used, the data and the CDO server will be part of the company infrastructure. Then, they should be protected in a similar way to other assets (databases, web applications…) of the company infrastructure.

- **About External tools**

  From the Platform (Help menu) and the User Manual, there must exist information about what are the supported external features that do not provide seamless interoperability means with AMASS. Moreover, the supported high-level requirements that are only provided by those external tools must be clearly identified. In general, these tools can be used under proprietary license. So, the interfaces between open and close source must be clearly defined. In addition, it must be made available for the

user the necessary procedure(s) to have the tool(s) installed and connected with AMASS platform as well as a documentation material.

- **About Intellectual Property**

  The AMASS results are protected by the Consortium Agreement to protect knowledge generated within the project that will be released to the AMASS open community. These results are also publicly available on open access publications; hence, authorship and copyrights are already handled in these results, e.g. by the publisher for the publications. The open platform itself is release as an Eclipse open source tool, so it is under the Eclipse Public License v2.0, and possibly the Creative Commons by-sa 4.0 license for documentation. For the specific case of the invention of a new method, a patent application may be investigated with some collaboration agreement establishment to let business users exploit it. But as software patent is not possible, new technology inventions may be protected by others means by the involved partners, e.g. through a spin-off company or a tool license.

8. AMASS Future Exploitation Perspectives

From an industrial perspective, the AMASS project provides a platform that fulfils many requirements. It is open, easy to extend, and sustainable (by means of re-using and recycling knowledge). As with many large applications, companies will likely suffer from drawbacks initially during the phase when the platform is established (similar to enterprises establishing Enterprise Resource Planning systems from Oracle/SAP). We assume though that once the Platform is established, it will sustainably contribute and pay off in the long run. For instance, when employees with expert knowledge leave a company, they can leave valuable information thoroughly categorized behind. Furthermore, the effort to harmonize standards from different domains decreases, and safety/security engineers from one domain find it easier to switch to another domain once they get familiar with similar standards from the same equivalence class. Analogous to the adaptation phase of companies to new ERP systems, it might be interesting to investigate how long it takes until it pays off to replace current V&V efforts with the AMASS Platform.

From an academic perspective, AMASS shows huge potential as a rich testbed for industrial applications that now might become more accessible for novel technologies and techniques, arching from testing methods from software engineering to formal verification. Two very interesting domains are: i) safety impact of security, and ii) certification and assurance of multi-critical probabilistic/stochastic processes. The first point targets assessing (functional) safety and security concerns alike (even in parallel), and the mutual effect one can have on the other. The second point targets the inclusion of reasoning beyond functional safety. When certifying a system, functional properties are important for the industry.

9. Conclusions

This report D2.9 provides an overview of the evaluation activities done around the AMASS Platform. The document includes:

- A short introduction to the ARTA with summarized information about the global AMASS methodology and the AMASS Platform tools, including an overview of the platform architecture, the relevant building blocks and functionalities.
- A summary of the outcomes and the limitations encountered during the validation campaigns executed on the AMASS Open Tool platform with respect to the high-level requirements elicited for the Platform at the beginning of the project.
- A usability evaluation of the AMASS Platform based on the analysis by the case studies owners of the tool chains they have been using on their applications.
- An evaluation of the tool qualification focusing on some tool chain provided by the AMASS Platform with regards to expectations from different standards criteria.
- An assessment of the TRL achieved by some key components of the Platform with the associated justifications.
- An evaluation of the AMASS public artefacts, mainly the AMASS websites and the User Manual, following the point of view of an external user, stranger to AMASS development activities.
- Some recommendations concerning the AMASS Platform further usage and evolution.
- Some tracks for future exploitation from industrial and academic perspectives.

As D2.9 presents some feedback on different evaluations of the AMASS Platform and companion artefacts, this report can serve as source of information to whoever would like to maintain and evolve the AMASS Platform.

Abreviations and Definitions

AESAS	Association of European Suppliers for Automotive Software
AMASS	Architecture-driven, Multi-concern and Seamless Assurance and Certification of Cyber-Physical Systems
API	Application Programming Interface
ARTA	AMASS Reference Tool Architecture
ASIL	Automotive Safety Integrity Level
AUTOSAR	AUTomotive Open System ARchitecture
BVR	Base Variability Resolution
CACC	Cooperative Adaptive Cruise Control
CACM	Common Assurance & Certification Metamodel
CCC	Correctness, Consistency and Completeness
CDO	Connected Data Objects
CHESS	Composition with Guarantees for High-integrity Embedded Software Components Assembly
CPS	Cyber Physical Systems
CPU	Central Processing Unit
CRUD	Create, Read, Update and Delete
CSD	Composite Structure Diagram
CVL	Common Variability Language
DSL	Domain Specific LanguageEPF Eclipse Process Framework
ERP	Enterprise Resource Planning
FLA	Failure Logic Analysis
FMVEA	Failure Modes, Vulnerabilities and Effect Analysis
FPTC	Failure Propagation Transform Calculus
FT&AT	Fault Tree & Attack Tree
FTA	Fault tree analysis
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
GB	Gigabyte
GUI	Graphical User Interface
HARA	Hazard Analysis and Risk Assessment
HAZOP	HAZard and OPerability analysis
HMI	Human Machine Interface
IMA	Integrated Modular Avionics
IBD	Internal Block Diagram
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
KM	Knowledge Management
MARTE	Modelling and Analysis of Real Time and Embedded systems
NASA	National Aeronautics and Space Administration

NuSMV	New Symbolic Model Verifier (a symbolic model checker tool for finite state systems)
OCRA	Othello Contracts Refinement Analysis
OMG	Object Management Group
OPENCOSS	Open Platform for EvolutioNary Certification Of Safety-critical Systems
OS	Operating System
OSLC	Open Services for Lifecycle Collaboration
OT&E	Operational Test & Evaluation
RAM	Random-Access Memory
RIA	Research and Innovation Action
RQA	Requirement Quality Analyser
SPEM	Software & Systems Process Engineering Metamodel
SUS	System Usability Scale
STO	Scientific and Technical Objective
SysML	Systems Modelling Language
UML	Unified Modelling Language
URL	Uniform Resource Locator
TARA	Threat Analysis and Risk Assessment
TCL	Tool Confidence Level
TD	Tool error Detection
TI	Tool Impact
TQL	Tool Qualification Level
TRL	Technology Readiness Level
UI	User Interface
V&V	Verification & Validation
VPN	Virtual Private Network
WBS	Work Break Down Structure
WP	Workpackage
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSAP	Symbolic model checking tool for safety assessment of synchronous finite-state and infinite-state systems

References

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

[2] SafeCer Project, 2015. (Certification of Software-intensive Systems with Reusable Components) http://cordis.europa.eu/project/rcn/103721_en.html and http://cordis.europa.eu/project/rcn/105610_en.html

[3] PolarSys. https://www.polarsys.org

[4] AMASS D2.5 AMASS User guidance and methodological framework, November 2018

[5] AMASS Platform User Manual[32]. D2.5 AMASS User guidance and methodological framework, November 2018.

[6] AMASS Platform Developers Guide[33]. D2.5 AMASS User guidance and methodological framework, November 2018.

[7] AMASS D2.4 AMASS reference architecture (c), June 2018.

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

[9] CHESS project. http://www.chess-project.org/

[10] OMG - Semantics of Business Vocabulary and Rules™ (SBVR™) version 1.3, 2015 http://www.omg.org/spec/SBVR/1.3

[11] WEFACT. http://www.ait.ac.at/en/research-fields/verification-validation/methods-and-tools/wefact/

[12] Eclipse Process Framework Project. https://eclipse.org/epf/

[13] OSLC. http://open-services.net/specifications/

[14] OSLC-KM for Knowledge Management http://trc-research.github.io/spec/km/, Llorens, J., Morato, J., Genova, G., Fuentes, M., Quintana, V., & Díaz, I. (2004). RHSP: An information representation model based on relationship. Studies in fuzziness and soft computing, 159, 221-253.

[15] Papyrus Eclipse project. https://eclipse.org/papyrus/

[16] Capra project. https://projects.eclipse.org/proposals/capra

[17] AUTomotive Open System Architecture. http://www.autosar.org

[18] Gaska, T., Watkin, C., & Chen, Y. (2015). Integrated modular avionics-past, present, and future. *IEEE Aerospace and Electronic Systems Magazine*, *30* (9), 12-23.

[19] Eclipse Process Framework Project. https://eclipse.org/epf/

[20] AMASS D2.7 Integrated AMASS platform (b), January 2018

[21] AMASS Platform bundle, 2018 https://www.polarsys.org/opencert/

[22] CONCERTO Deliverable D3.3 November 2015 Design and implementation of analysis methods for non-functional properties – Final version

[23] Medini Analyzer. https://www.ansys.com/fr-fr/products/systems/ansys-medini-analyze

[24] Safety Architect. https://www.all4tec.net/documentation-safety-architect

[25] Cyber Architect. https://www.all4tec.net/documentation-cyber-architect

[26] Sabotage. https://www.cyberssbytecnalia.com/node/271

[27] SAVONA. https://www.assystem-germany.com/en/products/savona/

[28] Verification Studio. https://www.reusecompany.com/verification-studio

[32] The AMASS Platform User Manual has been included in Annex A of D2.5 AMASS User guidance and methodological framework.

[33] The AMASS Platform Developers Guide has been included in Annex B of D2.5 AMASS User guidance and methodological framework.

[29] Mankins, J. C.. Technology readiness assessment: A retrospective. Acta Astronautica 65(9- 10), pp.1216-1223. 2009

[30] ISO 16290:2014, Space systems -- Definition of the Technology Readiness Levels (TRLs) and their criteria of assessment, 2014

[31] Julio Cesar Lemos, Milton Freitas Chagas, "Application of maturity assessment tools in the innovation process: converting system's emergent properties into technological knowledge",RAI Revista de Administração e Inovação, Volume 13, Issue 2, 2016

[32] Standard IEC 61508 "Functional safety of electrical/electronic/programmable electronic safety-related systems".

[33] ED-215 (DO-330) "Software Tool Qualification Considerations".

[34] Standard ISO 26262 "Road vehicles – Functional safety".

[35] AMASS D4.6 Prototype for multiconcern assurance (c), August 2018.

[36] AMASS D3.6 Prototype for architecture-driven assurance (c), August 2018.

[37] Standard EUROCAE ED-12C "Software Considerations in Airborne Systems and Equipment Certification".

[38] AMASS D2.6 Integrated AMASS platform (a), November 2017

[39] AMASS D2.8 Integrated AMASS platform (c), December 2018

[40] Ronan Barrett, Francis Bordeleau, "5 Years of 'Papyrusing' – Migrating Industrial Development from a Proprietary Commercial Tool to Papyrus", OSS4MDE@MoDELS 2015: 3-12

[41] Bordeleau, Francis. "Model-Based Engineering: A New Era Based on Papyrus and Open Source Tooling." *OSS4MDE@MoDELS* (2014).

[42] Morayo Adedjouma, Thibaud Thomas, Chokri Mraidha, Sebastien Gerard, Guillaume Zeller, "From Document-Based to Model-Based System and Software Engineering: Experience Report of a Selective Catalytic Reduction System Development", OSS4MDE@MoDELS 2016: 27-36

[43] AMASS D3.3 Design of the AMASS tools and methods for architecture-driven assurance (b), March 2017

[44] AMASS D4.3 Design of the AMASS tools and methods for multiconcern assurance (b), April 2018

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

[46] AMASS D6.3 Design of the AMASS tools and methods for cross/intra-domain reuse (b), July 2018

[47] AMASS D1.4 AMASS demonstrators (a), April 2017

[48] AMASS D1.5 AMASS demonstrators (b), March 2018

[49] AMASS D1.1 Case studies description and business impact, May 2018

[50] Jakob Nielsen and Rolf Molich. 1990. Heuristic evaluation of user interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '90), Jane Carrasco Chew and John Whiteside (Eds.). ACM, New York, NY, USA, 249-256.

[51] AMASS D3.8 Methodological guide for architecture-driven assurance (b), October 2018.

[52] AMASS D4.8 Methodological guide for cross/intra-domain reuse (b), October 2018.

[53] AMASS D5.8 Methodological guide for seamless interoperability (b), October 2018.

[54] AMASS D6.8 Methodological guide for cross/intra-domain reuse (b), November 2018.

[55] Ougier, François & Terrier, François. (2019). EDONA: an Open Integration Platform for Automotive Systems Development Tools.

[56] Ansgar Radermacher, Brahim Hamid, Manel Fredj, and Jean-Louis Profizi. 2015. Process and tool support for design patterns with safety requirements. In Proceedings of the 18th European Conference on Pattern Languages of Program (EuroPLoP '13).

[57] Muhammad Atif Javed and Barbara Gallina, "Get EPF Composer back to the future: A trip from Galileo to Photon after 11 years", EclipseCon, Toulouse, France, JUNE 13 - 14, 2018

[58] BVR Tool, https://github.com/SINTEF-9012/bvr, accessed: 2019-03-11.

[59] EPF Composer 1.5.2, https://www.eclipse.org/epf/downloads/tool/epf1.5.0_downloads.php

[60] Bugzilla – Bug 516608, Upgrade to newer version of Eclipse and to Java 8, https://bugs.eclipse.org/bugs/show_bug.cgi?id=516608

[61] Lewis J.R., Sauro J. (2009) The Factor Structure of the System Usability Scale. In: Kurosu M. (eds) Human Centered Design. HCD 2009. Lecture Notes in Computer Science, vol 5619. Springer, Berlin, Heidelberg

[62] AMASS D1.6 AMASS demonstrators (c), March 2019

Appendix A: Coverage of High Level Requirements by the AMASS Platform

Table 16. Coverage of High-Level Requirements related to the AMASS Platform Basic Building Blocks

High Level Requirements related to AMASS Platform Basic Building Blocks		Campaign[34]	Validation results
1.-High Level Requirements for System Component Specification			
WP3_SC_001	System abstraction levels browsing	#1	Passed
WP3_SC_002	System abstraction levels editing	#1	Passed
WP3_SC_003	Modelling languages for component model		Cancelled
WP3_SC_004	Formalize requirements into formal properties	#1	Passed
WP3_SC_005	Requirements allocation	#1	Passed
WP3_SC_006	Specify component behavioural model (state machines)	#1	Passed
WP3_SC_007	Fault injection (include faulty behaviour of a component)	#2	Passed
2.-High Level Requirements for Assurance Case Specification			
WP4_ACS_001	Assurance case edition	#3	Passed
WP4_ACS_002	Argumentation architecture	#2	Passed
WP4_ACS_003	Drag and drop argumentation patterns	#2	Passed but
WP4_ACS_004	Provide guidelines for argumentation patterns	#3	Passed
WP4_ACS_005	Provide a structured language to the text inside the claims	#3	Failed
WP4_ACS_006	Provide guidelines for argumentation	#3	Passed
WP4_ACS_007	Argumentation import/export	#2	Failed
WP4_ACS_008	Traceability of the dependability case	#3	Passed
WP4_ACS_009	Find high level claims		Cancelled
WP4_ACS_010	Composition of the overall argument	#3	Passed but
WP4_ACS_011	Assurance case status report	#3	Passed
WP4_ACS_012	Formal validation of assumptions and context when arguments modules are connected		Cancelled
WP4_ACS_013	Provide quantitative confidence metrics about an assurance case in a report	#3	Passed
3.-High Level Requirements for Evidence Management			
WP5_EM_001	Evidence characteristics specification	#1	Passed
WP5_EM_002	Evidence traceability	#1	Passed
WP5_EM_003	Evidence change impact analysis	#1	Passed
WP5_EM_004	Evidence evaluation	#1	Passed
WP5_EM_005	Evidence information import	#1	Passed
WP5_EM_006	Evidence information export	#2	Passed
WP5_EM_007	Derivation of evidence characterization model	#3	Passed
WP5_EM_008	Visualization of chains of evidence	#3	Passed
WP5_EM_009	Suggestion of evidence traces	#3	Passed
WP5_EM_010	Evidence lifecycle information storage	#1	Passed

[34] We reference the final campaign in which the feature has been validated regardless if it has been evaluated in previous campaigns.

WP5_EM_011	Interactive evidence change impact analysis	#1	Passed
WP5_EM_012	Evidence trace verification	#3	Passed
WP5_EM_013	Link of evidence to other assets	#1	Passed
WP5_EM_014	Evidence resource specification	#1	Passed
WP5_EM_015	Resource part selection	#3	Passed
WP5_EM_016	Evidence report generation	#3	Passed
4.-High Level Requirements for Compliance Management			
WP6_CM_001	Modelling of standards	#2	Passed but
WP6_CM_002	Tailoring of Standards models to specific projects	#1[35]	Passed
WP6_CM_003	Correlating processes to the requirements	#3	Passed
WP6_CM_004	Triggering compliance Checking	#3	Passed
WP6_CM_005	Compliance Monitoring	#1	Passed
WP6_CM_006	Compliance Status to Externals	#3	Passed
WP6_CM_007	Useful Feedback Upon Violations	#3	Passed
WP6_CM_008	Process Compliance (informal) management	#1	Passed
WP6_CM_009	Process Compliance (formal) management)	#3	Passed
WP6_CM_010	Compliance map generation from argument evidences	#3	Passed
5.-High Level Requirements for Access Manager			
WP5_AM_001	User authentication	#3	Passed
WP5_AM_002	User access	#3	Passed
WP5_AM_003	User action log	#3	Passed
WP5_AM_004	User profiles	#3	Passed
WP5_AM_005	Access rights groups	#3	Passed
6.-High Level Requirements for Data Manager			
WP5_DM_001	Multi-platform availability	#3	Passed
WP5_DM_002	Simultaneous data access	#3	Passed
WP5_DM_003	Consistent data access	#3	Passed
WP5_DM_004	Real-time data access feedback	#3	Passed
WP5_DM_005	System artefact information storage	#3	Passed
WP5_DM_006	Standard formats storage	#3	Passed
WP5_DM_007	Data versioning	#3	Passed
WP5_DM_008	Secure data access	#3	Passed

[35] The feature has been improved during Campaign #3 to support 1) the transformation of process from EPF Composer to OpenCert process and evidence models, 2) the transformation of Standard's Requirement from EPF Composer to OpenCert Baseline Model

Table 17. Coverage of High-Level Requirements related to Architecture-Driven Assurance (STO1)

High Level Requirements related to Architecture-Driven Assurance (STO1)		Campaign	Validation results
1.-High Level Requirements for System Architecture Modelling for Assurance			
WP3_SAM_001	Trace component with assurance assets	#1	Passed
WP3_SAM_002	Impact assessment if the component changes	#3	Passed but
WP3_SAM_003	Compare different architectures according to different concerns which have been specified before	#3	Passed
WP3_SAM_004	Integration with external modelling tools	#3	Passed
2.-High Level Requirements for Assurance Patterns Library Management			
WP3_APL_001	Drag and drop an architectural pattern	#3	Passed
WP3_APL_002	Edit an architectural pattern	#3	Passed
WP3_APL_003	Use of architectural patterns at different levels	#3	Passed
WP3_APL_004	Architectural Patterns suggestions		Cancelled
WP3_APL_005	Generation of argumentation fragments from architectural patterns/decisions	#3	Passed
3.-High Level Requirements for Contract Based Assurance Composition			
WP3_CAC_001	Validate composition of components by validating their assurance contract	#3	Passed
WP3_CAC_002	Assign contract to component	#1	Passed
WP3_CAC_003	Structure properties into contracts (assumptions/guarantees)	#1	Passed
WP3_CAC_004	Specify contract refinement	#1	Passed
WP3_CAC_005	General management of contract component assignments	#3	Passed
WP3_CAC_006	Refinement-based overview	#2	Passed
WP3_CAC_007	Overview of check refinements results	#2	Passed
WP3_CAC_008	Contract-based validation and verification	#3	Passed
WP3_CAC_009	Improvement of Contract definition process	#2	Passed
WP4_CAC_010	Contract-based trade-off analysis	#3	Passed
WP3_CAC_011	Overview of contract-based validation for behavioural models	#3	Passed
WP3_CAC_012	Browse Contract status	#1	Passed
WP3_CAC_013	Specify contracts defining the assumption and the guarantee elements	#1	Passed
4.-High Level Requirements for V&V Based Assurance			
WP3_VVA_001	Traceability between different kinds of V&V evidence	#1	Passed
WP3_VVA_002	Trace model-to-model transformation	#3	Passed
WP3_VVA_003	Validate requirements checking consistency, redundancy, … on formal properties	#3	Passed
WP3_VVA_004	Trace requirements validation checks	#3	Passed
WP3_VVA_005	Verify (model checking) state machines	#3	Passed
WP3_VVA_006	Automatic provision of HARA/TARA-artifacts	#3	Passed
WP3_VVA_007	Generation of reports about system description/verification results	#3	Passed
WP3_VVA_008	Automatic test cases specification from assurance requirements specification		Cancelled
WP3_VVA_009	Capability to connect to tools for test case generation based on assurance requirements specification of a component/system		Cancelled
WP3_VVA_010	Model-based safety analysis	#3	Passed
WP3_VVA_011	Simulation-based Fault Injection	#3	Passed
WP3_VVA_012	Design Space Exploration	#3	Passed

Table 18. Coverage of High-Level Requirements related to Multi-Concern Assurance (STO2)

High Level Requirements related to Multi-Concern Assurance (STO2)		Campaign	Validation results
1.-High Level Requirements for Dependability Assurance Modelling			
WP4_DAM_001	Capability to model relationships between concerns	#2	Passed
WP4_DAM_002	Capability to capture conflicts occurring during system development and the trade-off process	#2	Passed
2.-High Level Requirements for Contract Based Multi-concern Assurance			
WP4_CMA_001	The AMASS tools must support specification of variability at the argumentation level	#3	Passed
WP4_CMA_002	Component contracts must support multiple concerns	#3	Passed
WP4_CMA_003	Contract based multi-concern assurance	#2	Passed
3.-High Level Requirements for System Dependability Co-Analysis/Assessment			
WP4_SDCA_001	System dependability co-architecturing and co-design	#3	Passed
WP4_SDCA_002	System dependability co-verification and co-validation	#3	Passed
WP4_SDCA_003	The system shall allow combinations of safety and security analysis	#3	Passed

Table 19. Coverage of High-Level Requirements related to Seamless Interoperability (STO3)

High Level Requirements related to Seamless Interoperability (STO3)		Campaign	Validation results
1.-High Level Requirements for Tool Integration Management			
WP5_TI_001	Automatic data collection	#3	Passed
WP5_TI_002	Automatic data export	#3	Passed
WP5_TI_003	Tool chain deployment support	#3	Passed
WP5_TI_004	System analysis tools interoperability	#3	Passed
WP5_TI_005	System specification tools interoperability	#3	Passed
WP5_TI_006	V&V tools interoperability	#3	Passed
WP5_TI_007	Version management tools interoperability	#3	Passed
WP5_TI_008	Quality management tools interoperability	#3	Passed
WP5_TI_009	MS Office applications interoperability	#3	Passed
WP5_TI_010	Interoperability throughout CPS lifecycle	#3	Passed
WP5_TI_011	Non-proprietary data exchange	#3	Passed
WP5_TI_012	Data entry effort	#3	Passed
WP5_TI_013	Continuous data management	#3	Passed
WP5_TI_014	Client-server support	#3	Passed
WP5_TI_015	Service offer and discovery	#3	Passed
WP5_TI_016	Performance monitoring	#3	Passed
WP5_TI_017	Standards-based interoperability	#3	Passed
WP5_TI_018	Extended standard-based interoperability	#3	Passed
2.-High Level Requirements for Collaborative Work Management			
WP5_CW_001	Collaborative system analysis	#3	Passed
WP5_CW_002	Collaborative system specification	#3	Passed
WP5_CW_003	Collaborative management of compliance with standards and of process assurance	#3	Passed
WP5_CW_004	Collaborative re-certification needs & consequences analysis	#3	Passed
WP5_CW_005	Collaborative system V&V	#3	Passed
WP5_CW_006	Collaborative model-based systems engineering	#3	Passed

WP5_CW_007	Collaborative assurance evidence management	#3	Passed
WP5_CW_008	Collaborative product reuse needs & consequences analysis	#3	Passed
WP5_CW_009	Collaborative assurance case specification	#3	Passed
WP5_CW_010	Collaborative compliance needs specification	#3	Passed
WP5_CW_011	Collaborative assurance assessment	#3	Passed
WP5_CW_012	Collaborative compliance assessment	#3	Passed
WP5_CW_013	Metrics & measurements reports	#3	Passed
3.- High Level Requirements for Tool Quality Assessment and Characterization			
WP5_TQ_001	Tool qualification information needs	#3	Passed
WP5_TQ_002	Tool quality evidence management	#3	Passed
WP5_TQ_003	Tool quality information import	#3	Passed
WP5_TQ_004	Tool quality needs indication	#3	Passed
WP5_TQ_005	Tool quality requirements fulfilment	#3	Passed

Table 20. Coverage of High-Level Requirements related to Cross/Intra-Domain Reuse (STO4)

High Level Requirements related to Cross/Intra-Domain Reuse (STO4)		Campaign	Validation results
1.-High Level Requirements for Reuse Assistant (Cross/Intra-Domain)			
WP6_RA_001	Intra-Domain, Intra standard, Reuse Assistance	#3	Passed
WP6_RA_002	Intra-Domain, Cross standards, Reuse Assistance	#3	Passed
WP6_RA_003	Intra-Domain, Cross versions, Reuse Assistance	#3	Passed
WP6_RA_004	Cross-Domain Reuse Assistance	#3	Passed
WP6_RA_005	Intra-Domain, Intra standard, Different Stakeholders, Reuse/Integration Assistance	#3	Passed
WP6_RA_006	Reusable off the shelf components	#3	Passed
WP6_RA_007	Provision of metrics about process-related reuse (e.g., size of commonality)		Cancelled
WP6_RA_008	Provision of metrics about product-related reuse (e.g., size of commonality)		Cancelled
WP6_RA_009	Provision of metrics about assurance case-related reuse (e.g., size of commonality)		Cancelled
2.-High Level Requirements for Semantic Standards Equivalence Mapping			
WP6_SEM_001	Semantics-based mapping of standards	#3	Passed
3.-High Level Requirements for Product/Process/Assurance Case Line Specification			
WP6_PPA_001	The AMASS tools must support variability management at process level	#3	Passed
WP6_PPA_002	Semi-automatic generation of product arguments	#3	Passed
WP6_PPA_003	Semi-automatic generation of process arguments	#3	Passed
WP6_PPA_004	The AMASS tools must support management of variability at the component level	#3	Passed
WP6_PPA_005	The AMASS tools must support variability management at the assurance case level	#3	Passed

Appendix B: AMASS Platform Common Assurance & Certification Metamodel (CACM)

The file **AppendixB_AMASS-Platform-CACM.pdf** contains the Common Assurance & Certification Metamodel of the AMASS Tool Platform. The CACM aims to provide a common understanding between the different domains and concerns involved in the AMASS project.

The first draft version of the CACM specification was included in the D2.2 deliverable “AMASS Reference Architecture (a)”. In the D2.4 deliverable “AMASS Reference Architecture (c) [7], the version of the CACM specification at the time of submission of D2.4 was included. The final version of the CACM specification is provided in this Annex.

ECSEL Research and Innovation actions (RIA)

AMASS

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

AMASS Platform Common Assurance & Certification Metamodel CACM

Work Package:	WP2: Reference Architecture and Integration
Dissemination level:	PU = Public
Status:	Final
Date:	31st January 2019
Responsible partner:	A. López (TECNALIA)
Contact information:	angel.lopez@tecnalia.com
Document reference:	AMASS_Platform_CACM_WP2_TEC_V1.0

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

Contributors

Names	Organisation
A. Ruiz, A. Lopez, G. Juez, C. Martinez	TECNALIA Research & Innovation (TEC)
J. L. de la Vara, J. M. Álvarez, E. Parra	Universidad Carlos III de Madrid (UC3)
L. M. Alonso, B. López	The REUSE Company (TRC)
S. Puri	INTECS (INT)
I. Ayala, B. Gallina. M. A. Javed, F. UL Muram	Maelardalens Hoegskola (MDH)
H. Espinoza	Commissariat à L'energie Atomique et aux Energies Alternatives (CEA)
A. Debiasi	Fondazione Bruno Kessler (FBK)

Document History

Version	Date	Status	Author (Partner)	Remarks
V0.8	2016-11-30	First version (D2.2)	Alejandra Ruiz	1st iteration of CACM
V0.9	2018-05-11	Complete version (D2.4)	Angel López	2nd iteration of CACM
V1.0	2019-01-31	Final version (D2.9)	Angel López	3rd iteration of CACM

TABLE OF CONTENTS

List of Figures

Executive Summary

AMASS is developing an integrated and holistic approach and supporting tools for assurance and certification of Cyber-Physical Systems (CPS) by creating and consolidating the first European-wide open certification/qualification platform, ecosystem and community spanning the largest CPS vertical markets. The approach is driven by architectural decisions, including multiple assurance concerns such as safety, security, robustness and reliability. The main goal is to reduce time, costs and risks for assurance and (re)certification by extending the OPENCOSS [1] and SafeCer [2] approaches for evolutionary compositional certification and cross-domain reuse.

This document describes the Common Assurance and Certification Metamodel (CACM). CACM aims to provide a common understanding between the different domains and concerns involved in the AMASS project.

A conceptual model does not reflect the right data to be exchanged and it is not a good interoperability enabler, because every tool provider will implement it in different way. This is the reason why this document also describes the implementation-based version of the CACM and the main differences between both.

The first draft version of the CACM specification was included in the D2.2 deliverable “AMASS Reference Architecture (a)” [3]. In the D2.4 deliverable [4], the actual version of the CACM specification at the time of submission of D2.4 was included. The final version of the CACM specification is provided as an Annex of the deliverable D2.9 “AMASS platform validation” [5].

1. Introduction

In this section, we present the metamodel created for AMASS, named the Common Assurance & Certification Metamodel (CACM) and the definitions of the concepts that includes. The metamodel is presented below in a series of "views", each of which represents a self-contained aspect of the whole. The general objectives and relationships of the views are captured in Figure 1. Note that there is not a 1:1 relationship between the views introduced here and the metamodels presented in the following sections, although the overall intent of the models accords with that in Figure 1. The metamodels are presented in sections below.

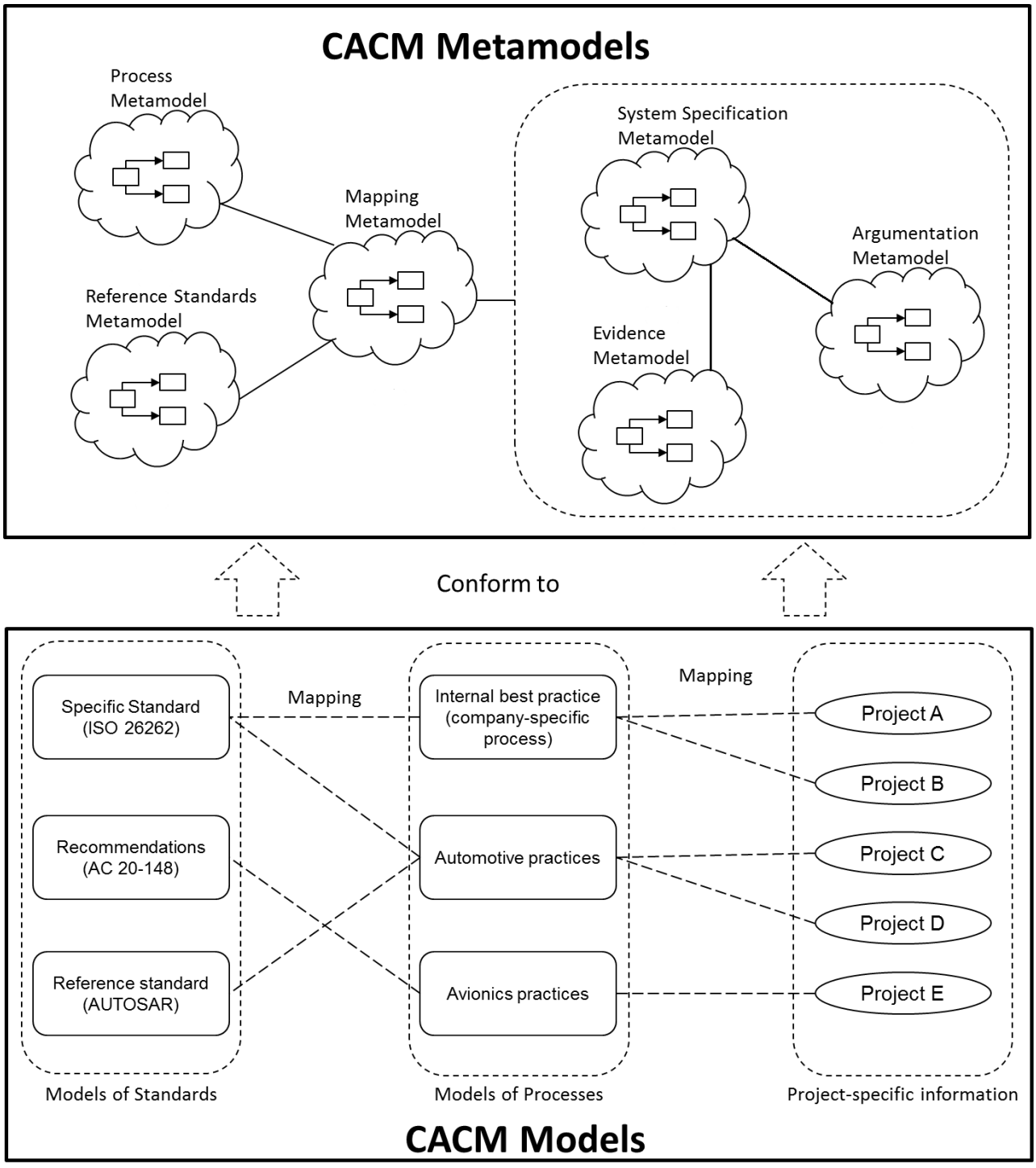


Figure 1. View of the CACM metamodels

As shown in Figure 1, the CACM metamodels comprise two types of conceptual aspects: project-specific aspects (System Specification Metamodel, Evidence Metamodel, and Argumentation Metamodel) and project-independent aspects (Reference Standards Metamodel, Process Metamodel, and Mappings Metamodel). When following this structure, the assurance assets from a project are mapped to a generic framework that reflects an abstract understanding of assurance.

In more specific terms:

- The **Reference Standards Metamodel** supports the specification of the main compliance criteria that have or might have to be considered in an assurance project. These criteria are usually represented by means of requirements to fulfil and recommendations based on criticality levels. The criteria can be specified from specific standards, recommended practices, or company-specific practices, and usually have to be tailored to project-specific characteristics.
- The **Process Metamodel** supports the specification of the process-specific compliance needs that have or might have to be considered in an assurance project. Such needs include not only the activities to execute, but also e.g. artefacts to manage. A process model represents the interpretation of how to comply with an assurance standard by following a specific process.
- By using the **Mappings Metamodel**, maps can be created to specify: (1) the degree of equivalence between the assurance information gathered during a project (e.g., artefacts) and a process model to declare compliance, and; (2) the degree of equivalence between models of standards or of process, or between models of standards and models of processes. The latter is a key for assurance reuse across different standards and domains. In general, the mappings aim to allow engineers and managers to make informed decisions about the appropriateness and implications of reusing assurance information across projects, safety standards, and domains.
- The **Evidence Metamodel** is targeted at recording the information related to the specific artefacts managed in an assurance project and that can be used as evidence of compliance or as evidence in an assurance case.
- The **System Specification Metamodel** supports the specification of system-specific details and decisions such a system's architecture and its component contracts.
- The **Argumentation (Assurance Case) Metamodel** is used to justify key safety-related decisions taken during the project or assurance decisions in general.

The following sections describe the different metamodels specified for the final version of the CACM from the Conceptual and Implementation perspective, and the maps between them.

2. Conceptual CACM

2.1 General Metamodel

2.1.1 Scope and Purpose

The General Metamodel factorises many metaclasses which are shared by various other metamodels in CACM.

2.1.2 Conceptual General Metamodel

The class diagram for the General Metamodel is presented in the figure below.

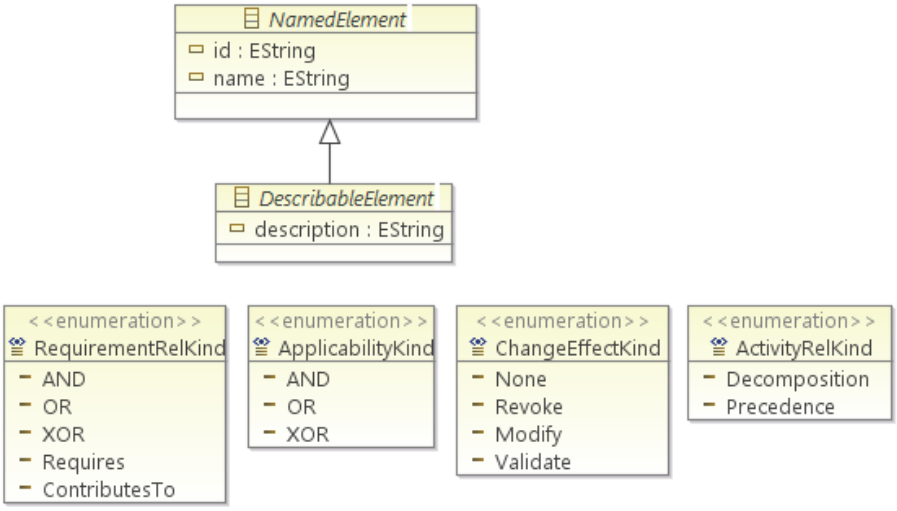


Figure 2. General Metamodel

2.1.2.1 NamedElement (abstract)

This class corresponds to the classes of the CACM metamodels for which an ID and a name can be specified.

Attributes

- id: String
  The ID of the NamedElement
- name: String
  The name of the NamedElement

Semantics

A Named Element models an element that can have an ID string and a Name string.

2.1.2.2 DescribableElement (abstract)

This class corresponds to the classes of the CACM metamodels for which a description can be specified.

Superclass

- *NamedElement*

Attributes

- description: String
  The description of the describable element

Semantics
A Describable Element models an element that can have a Description string.

2.1.2.3 ActivityRelKind (enumeration)

This enumeration corresponds to the possible relationships that can exist between two (reference) activities.

Literals

- Decomposition
  An activity is decomposed into several sub-activities.
- Precedence
  The execution of an activity precedes the execution of another.

2.1.2.4 ChangeEffectKind (enumeration)

This enumeration corresponds to the possible effects that a change in some (reference) artefact can have in a related (reference) artefact.

Literals

- None
  A change has no effect.
- Revoke
  A change causes revocation.
- Modify
  A change causes the need for some modification.
- Validate
  Some validation is necessary to determine the effect of a change.

2.1.2.5 RequirementRelKind (enumeration)

This enumeration corresponds to the possible relationships that can exist between two requirements.

Literals

- AND
  Both requirements must be fulfilled.
- OR
  At least one of the requirements must be fulfilled.
- XOR
  Only one of the requirements can be fulfilled.
- Requires
  The fulfilment of one requirement depends on the fulfilment of another requirement.
- Contributes To
  Fulfilment of a requirement contributes to the fulfilment of another. This relationship also implies that the former requirements corresponds a decomposition of the latter. It is the opposite relationship to “refined to”.

2.1.2.6 ApplicabilityKind (enumeration)

This enumeration corresponds to the possible relationships that can exist between two applicability specifications.

Literals

- AND
  Both applicability specifications must be fulfilled.
- OR
  At least one of the applicability specifications must be fulfilled.

- XOR
  Only one of the applicability specifications can be fulfilled.

2.1.3 Conceptual Property Metamodel

The Property Metamodel defines model elements to represent various statements related to the fundamental properties of assurance assets. The properties have a data type, a measurement unit and a value. The Property Value model element has not been defined in this metamodel, since it belongs to the assurance asset annotated.

The class diagram for the **Property Metamodel** is presented in the figure below.

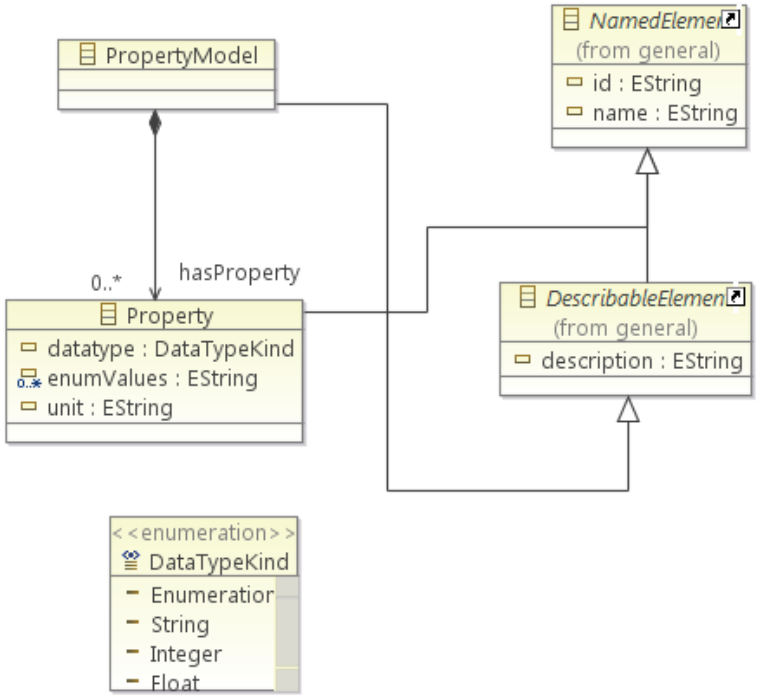


Figure 3. Property Metamodel

2.1.3.1 PropertyModel

This class corresponds to the model of properties which can be part of an assurance project.

Superclass

- *DescribableElement*

Relationships

- hasProperty: *Property* [0..*]
  The set of Properties that are part of the PropertyModel

Semantics

A Property Model represents the root model element to create properties.

2.1.3.2 Property

This class corresponds to a fundamental property of assurance assets.

Superclass

- *NamedElement*

Attributes

- datatype: *DataTypeKind*

The type of the data used to represent the values of a Property.

- enumValues: *String*
  The list of values modelled as strings (separated by a comma) which belong to the value space of Enumeration data type
- unit: *String*
  The measurement unit corresponding to the property values.

Semantics

A Property models a fundamental property of Assurance Assets such as an Artefact. The properties have a data type, a measurement unit and a value. The Property Value model element has not been defined in this metamodel, since it belongs to the assurance asset annotated. See for instance the Value metaclass in 3.4.3.5.

2.1.3.3 DataTypeKind (enumeration)

This enumeration corresponds to types of property values.

Literals

- Enumeration
  The value space characterized for a list of qualitative values.
- String
  The value space characterized by a string.
- Integer
  A value space characterized by Integer numbers.
- Float
  A value space characterized by Real numbers.

2.2 System Component Metamodel

2.2.1 Scope and Purpose

This section illustrates the System Component MetaModel supporting Architecture-driven assurance (CMMA). The metamodel is a review of the SafeCer metamodel [22] and embeds results from the SEooCMM metamodel [23].

This metamodel basically provides general architectural entities commonly available in standard modelling languages, like SysML and AADL (Architecture Analysis & Design Language). In addition, it enables contract-based design approach and the links to the other parts of the CACM needed to support the architecture driven assurance approach.

This is an abstract metamodel, in the sense that it is used to elaborate the domain needs in an easier way (e.g. without the need to introduce and face with the complexity of an existing standard component metamodel, like UML). The concepts available in the system component metamodel will be made available to the modeller by using standard/existing metamodel(s) properly adapted, like the CHESS UML/SysML profile [20].

It is worth noting that the main goal of the CMMA is not to provide a unified metamodel for system component, failure behaviour specification, etc. but to identify the links between architectural-related entities and the other parts of the CACM (e.g. argumentation, evidences), so to provide the model-based support for the architecture driven assurance approach.

2.2.2 Conceptual Model Definition

2.2.2.1 Modelling out of context

This part of the metamodel concerns the constructs that can be used to model entities out of a given context.

2.2.2.1.1 Block Type

A BlockType (Figure 4) represents a reusable unit out-of-context, i.e., a collection of features that are constant regardless of the context in which it is used. It can be used to represent any kind of system entities, e.g. HW, SW, functional, human.

The *realize* relationships allows to model that a block implements the functionality provided by other (more abstract) blocks, e.g. to model function blocks realized through SW/HW blocks.

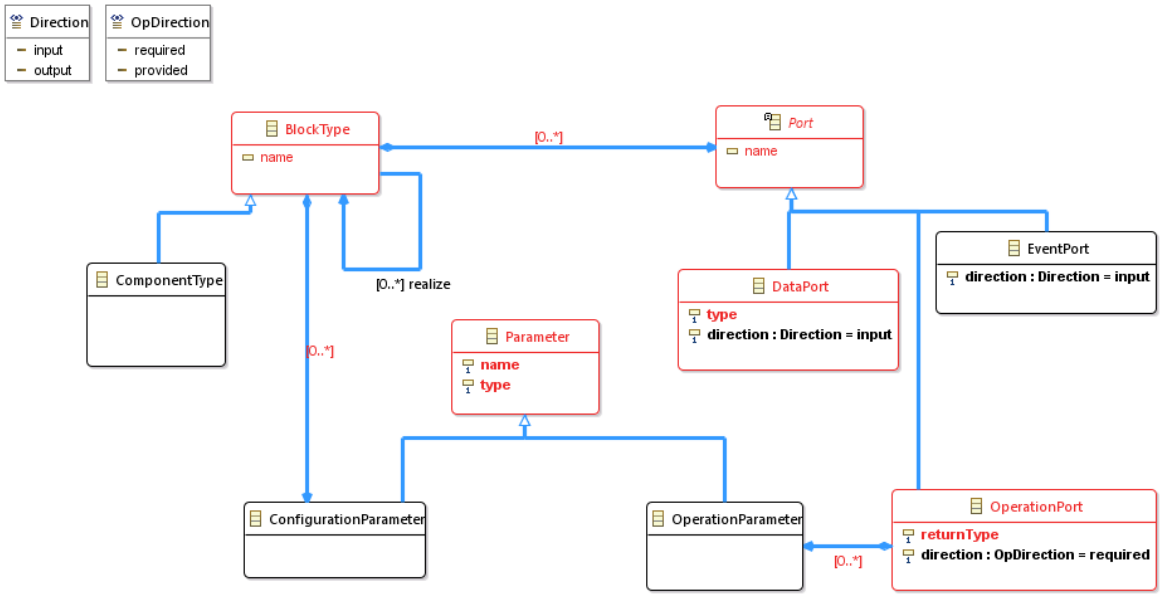


Figure 4. BlockType

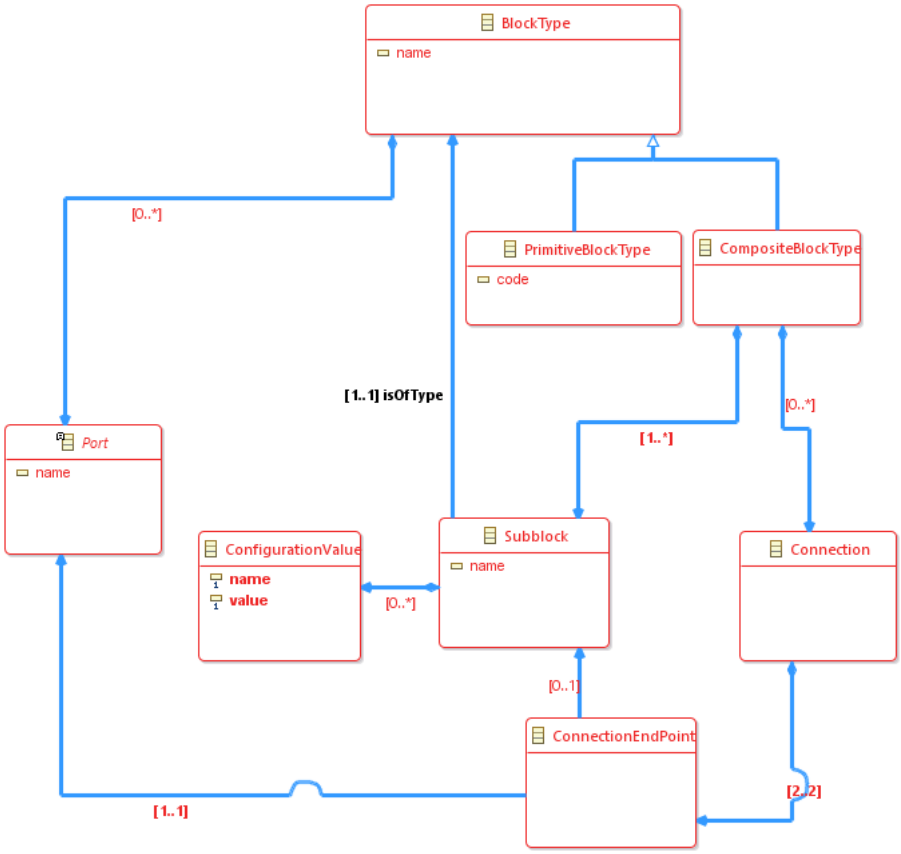


Figure 5. Composite BlockType

2.2.2.1.2 Port

A Port (Figure 4) represents an interaction point through which data can flow between the block and the context where it is placed. This is an abstract meta-class.

There are three types of ports: Data, Event and Operation ports. Each port also has a specified direction.

- **Data port**: A Data port is a point of interaction where typed data can be sent or received by the block. The direction of a data port is either output (sending data) or input (receiving data).
- **Event port**: An Event port is a point of interaction where events can be sent or received by the block. The direction of an event port is either output (sending events) or input (receiving events).
- **Operation port**: An Operation port is a point of interaction corresponding to a function or method, with a number of typed parameters and return type.

2.2.2.1.3 ConfigurationParameter

Configuration parameters (Figure 4) represent points of variability in a BlockType. They allow formulation of more detailed contracts by including them in assumptions or in the form of parametric contracts. For example, a contract could specify that the component requires at most 10+5*queue_length units of memory, where *queue_length* is one of the configuration parameters defined for the component type.

It can be set when the BlockType appears as Subblock or when it is instantiated (see BlockInstance) in a given system.

2.2.2.1.4 Subblock

A Subblock (Figure 5) represents a part of a decomposed BlockType. A Subblock is an occurrence of a given BlockType inside a parent BlockType.

Subblock is different from the BlockInstance concept (see BlockInstance) since Subblock is an occurrence of a BlockType in the context of a BlockType, while a BlockInstance is an occurrence of a BlockType in the context of a System.

When a composite BlockType is instantiated in a given system, its Subblocks are instantiated as well; in this way the Subblocks can be further configured at instance level.

Subblocks of the same parent BlockType can be connected together through ports. Also, Subblocks ports can be connected to the ports of the parent BlockType.

2.2.2.1.5 Connection and ConnectionEndPoint

Connection and ConnectionEndPoint (Figure 5) allow to connect Subblocks through the ports defined for the corresponding/typing BlockTypes.

2.2.2.2 Contracts

This part of the metamodel regards the constructs which enable contract-based design.

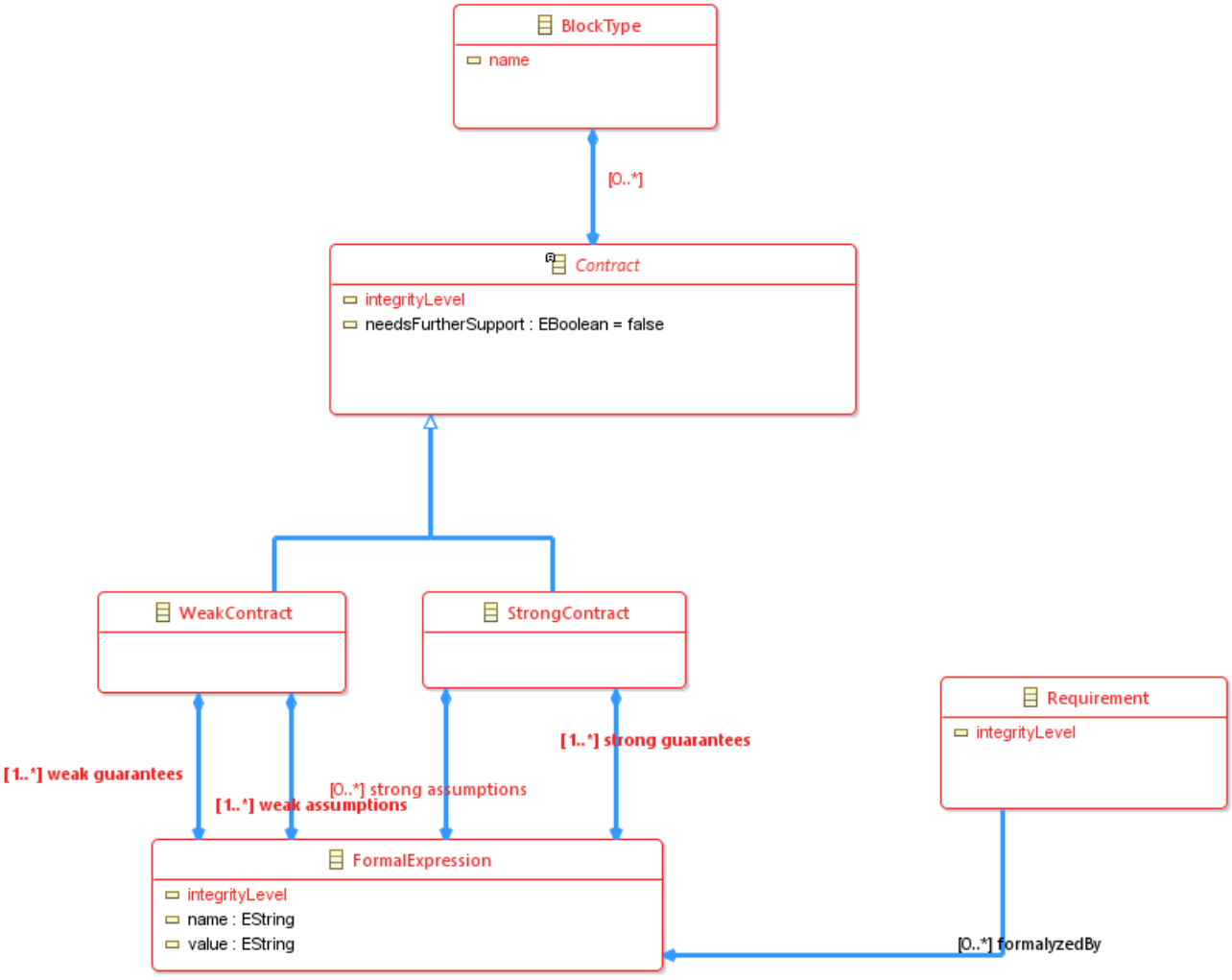


Figure 6. Contract

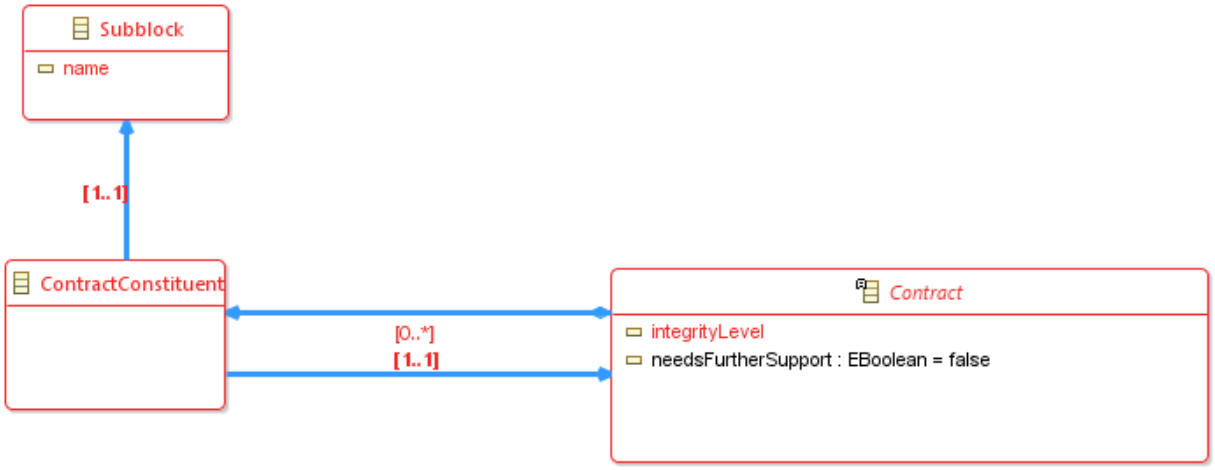


Figure 7. Contract refinement

2.2.2.2.1 Contract

Contracts (Figure 6) represent information about the block type, bundled together with explicit descriptions of the assumptions under which the information is guaranteed.
The general format of a contract can be defined as:

<A, G, {<B1, H1>, … , <Bn, Hn>}>

Where:

- *A* defines the strong assumptions that must hold in any context where the component type is used.
- *G* defines strong guarantees that always hold with no additional assumptions.
- *Bi* are weak assumptions that describe specific contexts where additional information is available.
- *Hi* are weak guarantees that are guaranteed to hold only in contexts where *Bi* hold.

A block type should never be used in a context where some strong assumptions are violated, but if some weak assumptions do not hold, it just means that the corresponding guarantees cannot be relied on.

Contract can have an *integrityLevel* stating the level of argumentation to be provided about the confidence in the contract; better semantic can be provided for integrity level according to adopted safety standard. Integrity level can be inherited by the Requirements associated to the Contract through *FormalExpression*.

The *needsFurtherSupport* Boolean attribute indicates if the contract is fully validated; if it is *false*, only partial evidence is provided with the contract and additional evidence should be provided.

2.2.2.2.2 FormalProperties

FormalProperty (Figure 6) represents a formalization of a requirement; it appears as assumption or guarantee of a Contract.

2.2.2.2.3 ContractConstituent

ContractConstituent (Figure 7) is used to model contract refinement along the BlockType decomposition, i.e. between BlockType and its parts (Subblocks).

E.g.: supposed to have contract C1 associated to BlockType B1, and B1 is decomposed into B1_1 and B1_2 Subblocks. B1_1 has contract C2 and B1_2 has contract C3 associated.

Then, ContractConstituent allows modelling that contract C1 is decomposed by the B1_1.C2 and B1_2.C3 contracts. In particular the “contract provided by a given Subblock” (e.g. B1_1.C2) is the kind of information stored in ContractConstituent.

2.2.2.3 Modelling in a given context

This part of the metamodel regards the constructs that can be used to model entities placed in a given context/system.

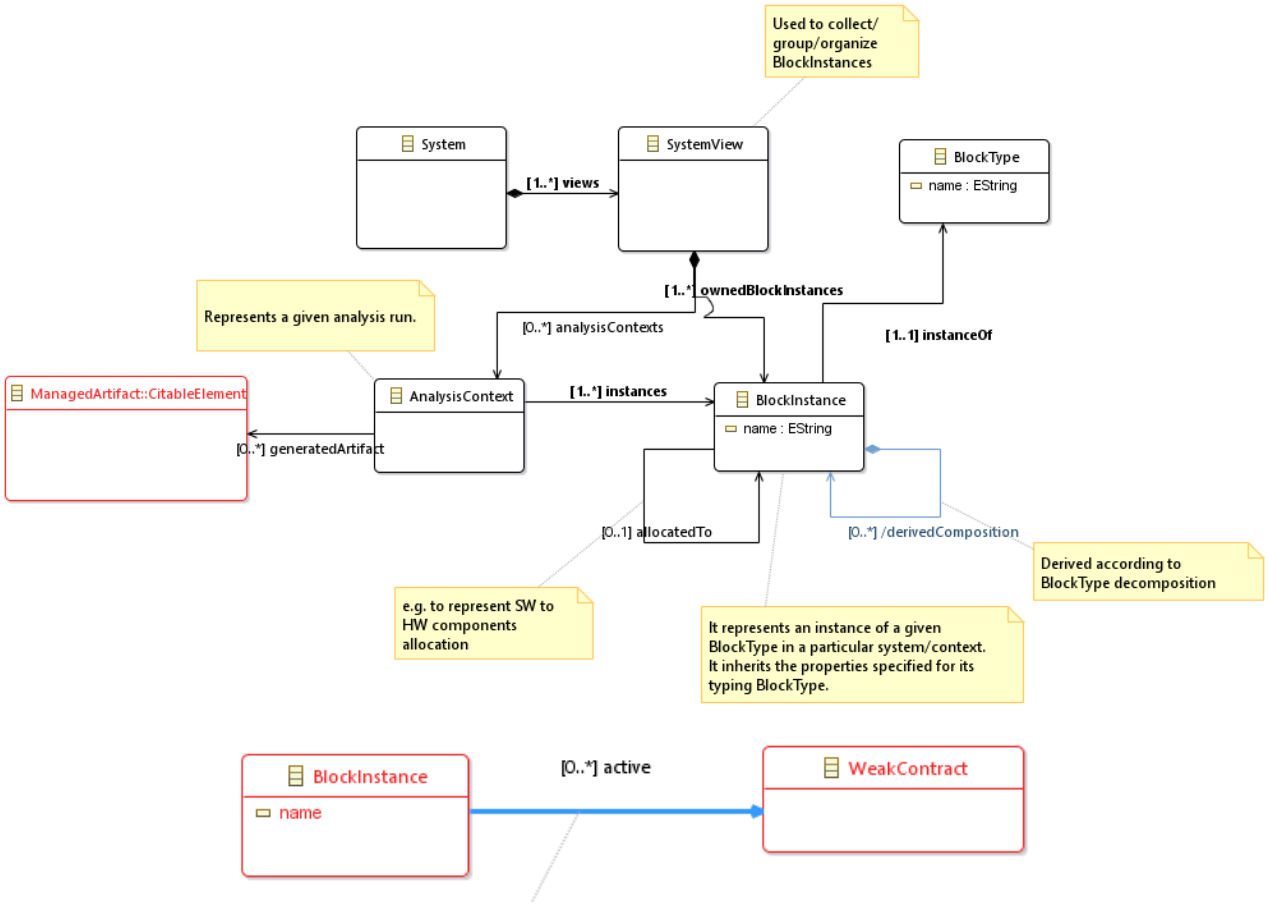


Figure 8. System

2.2.2.3.1 System

A System (Figure 8) represents a given cyber-physical system under design. It holds references to owned block instances through *software* and *platform* association; typically, the latter are created by instantiating a root composite BlockType.

2.2.2.3.2 BlockInstance

A BlockInstance (Figure 8) represents an instance of a given BlockType in a particular system/context; it inherits the properties (ports, parameters, contracts, subblocks) as specified for its typing BlockType. In particular, the decomposition structure defined for the typing BlockType is replicated at instance level through the *derivedComposition* link.

It has *allocatedTo* relationship to be used to model allocation of block instances, for instance like SW to HW instance blocks deployment.

The *active* link on the BlockInstance allows to specify the weak contracts associated to the typing BlockType which hold for a given block instance. Note that this can have impact on the modelled contract refinement. E.g., if a weak contract has been used to decompose a parent strong contract, then if the weak contract does not hold in a given context, then the contract refinement is invalid for that particular context.

A BlockInstance inherits the links to the evidence and assurance entities available for:

- the StrongContracts associated to the typing BlockType,
- the WeakContracts referred through the *active* relationships.

2.2.2.3.3 AnalysisContext

AnalysisContext (Figure 8) allows to represent a given analysis execution on a (sub)set of block instances. It has trace relationships to the artefacts produced by the corresponding analysis execution.

2.2.2.4 Failure Behaviour

This part of the metamodel regards the definition of the failure behaviour for a given BlockType. Basically, a BlockType can be decorated with a set of possible faults effecting the BlockType itself; the faults can be linked to failures of the given BlockType. Each failure can be described with a given FailureMode affecting the port of the BlockType.

It is expected that actual metamodels (e.g. provided by CHESS or Medini) will provide additional constructs to enrich the failure behaviours modelling (like about the impact of a failure mode on the nominal behaviours, quantitative value or qualitative expression for faults and failures occurrence).

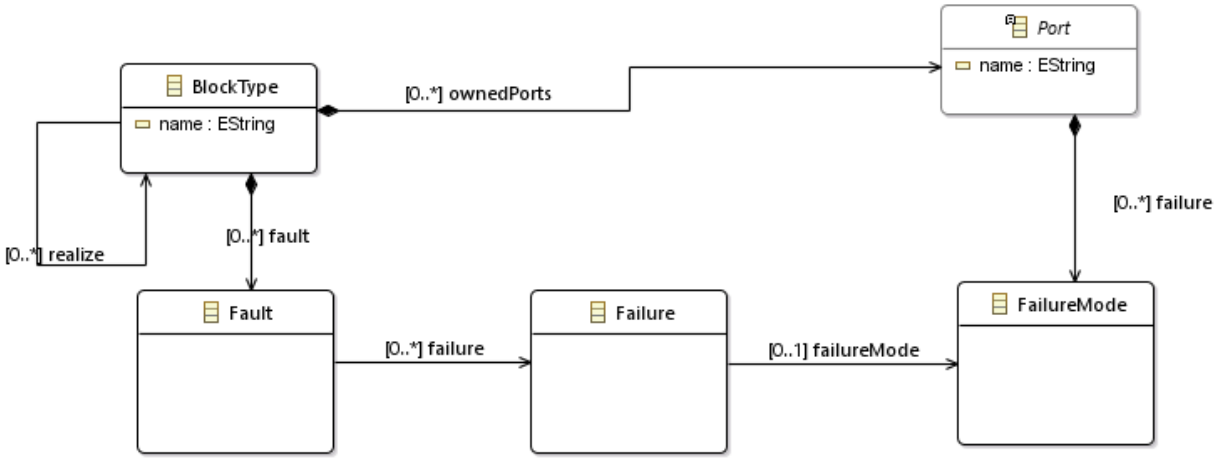


Figure 9. Failure Behaviour

2.2.2.5 Link to evidence and assurance cases

This part of the metamodel regards the connection to the assurance-related entities.

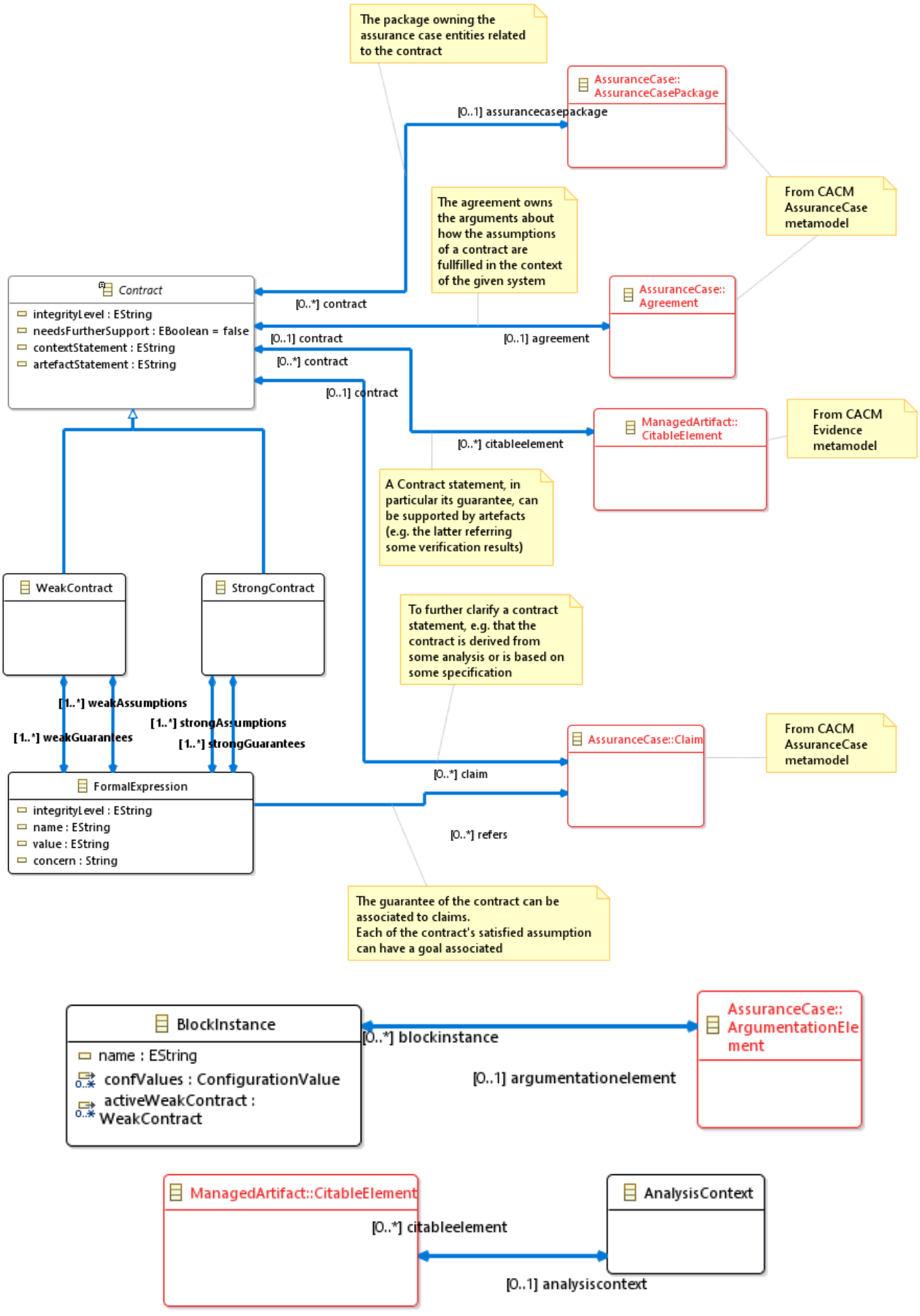


Figure 10. Artefact and assurance-related entities connections

2.2.2.5.1 CitableElement

Imported from AMASS CACM Managed Artifact Metamodel (see 2.4.3.2).

2.2.2.5.2 Claim

Imported from AMASS CACM Assurance Case Metamodel (see 3.3.2.10).

2.2.2.5.3 AssuranceCasePackage

Imported from AMASS CACM Assurance Case Metamodel (see 3.3.2.1).

2.2.2.5.4 Agreement

Imported from AMASS CACM Assurance Case Metamodel (see 3.3.2.2).

2.2.2.5.5 ArgumentationElement

Imported from AMASS CACM Assurance Case Metamodel (see 3.3.2.3).

2.2.2.5.6 BlockInstance

The BlockInstance entity (see 2.2.2.3.2) is extended with the following relationship:

- referenceArgumentation: ArgumentationElement
  - the arguments associated to the block instance

2.2.2.5.7 Contract

The Contract entity (see 2.2.2.2.1) is extended with the following relationships:

- assuranceCase: AssuranceCasePackage
  - the package(s) owning the assurance case entities related to the contract.
- agreement: Agreement
  - the agreement owns the arguments about how the assumption of a contract are fulfilled in the context of the given system.
- supportedBy: CitableElement
  - allows to model that a Contract statement, in particular its guarantees, can be supported by artefacts (e.g. the latter referring some verification results).
- claim: Claim
  - the referred claim allows to further clarify a contract statement; e.g. that the contract is derived from some analysis or is based on some specification.

The Contract entity is extended with the following attributes:

- contextStatement: String
  - store the informal description of what the contract means (which would be the context statement in the corresponding argumentation).
- artefactStatement: String
  - explain how a particular artefact relates to the contract.

2.2.2.5.8 FormalExpression

The FormalProperty entity (see 2.2.2.2.2) is extended with the following relationships:

- Refers: SupportStatement
  - Allows to map the guarantees of the contract to claims.
  - Allows to associate a claim (e.g. GSN away goal) to each of the contract's assumptions.

2.2.2.5.9 AnalysisContext

The AnalysisContext (see 2.2.2.3.3) is extended with a relationship to the artefacts produced by the corresponding analysis execution.

2.2.2.6 Link to executed process

This part of the metamodel regards the connection with the CACM executed process.

The subset of the CMMA entities that can play the role of input or output artefact for a given process activity is shown in Figure 11.

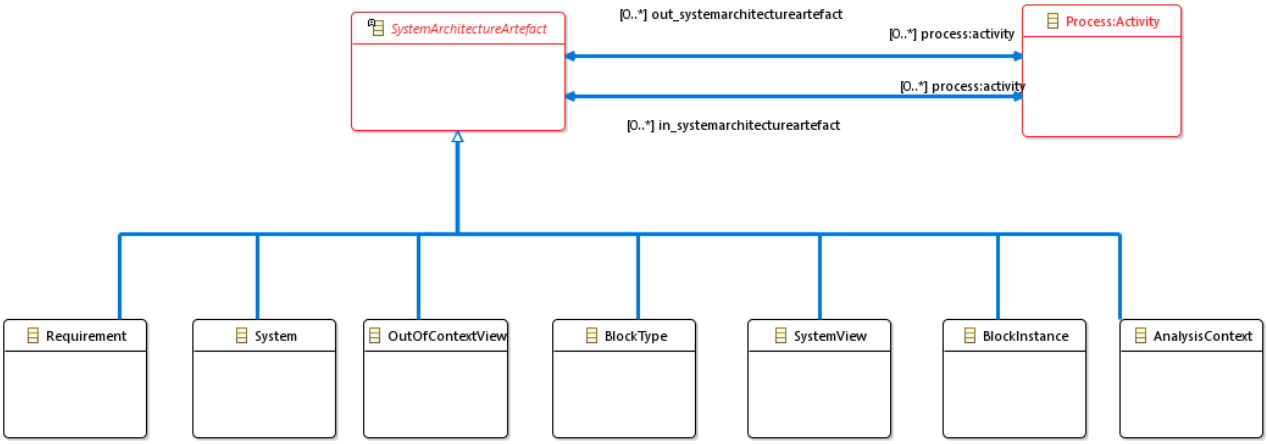


Figure 11. Links to the executed process

2.2.2.7 Instantiate a parameterized architecture

The InstantiatedArchitectureConfiguration (see 3.2.2.6) is associated to the System Block of the parameterized architecture, and it is used to store the information about one instantiated architecture, that is the list of parameters with their value and the reference to the System block of the instantiated architecture.

2.3 Assurance Case Metamodel

2.3.1 Scope and Purpose

The Assurance Case metamodel will describe how features of a generic assurance argument are linked together. The actual metamodel is being developed within WP4 and will be an extension to the argumentation sections of the existing OMG Structured Assurance Case Metamodel (SACM) [10]. Extensions are needed, as the current version of SACM does not provide enough support for patterns and variability arguments. These topics will be further analysed in WP3 (architectural patterns) and WP6 (variability).

2.3.2 Conceptual Model Definition

The metamodel presented here is an extension of the SACM Metamodel [10] from OMG (Object Management Group).

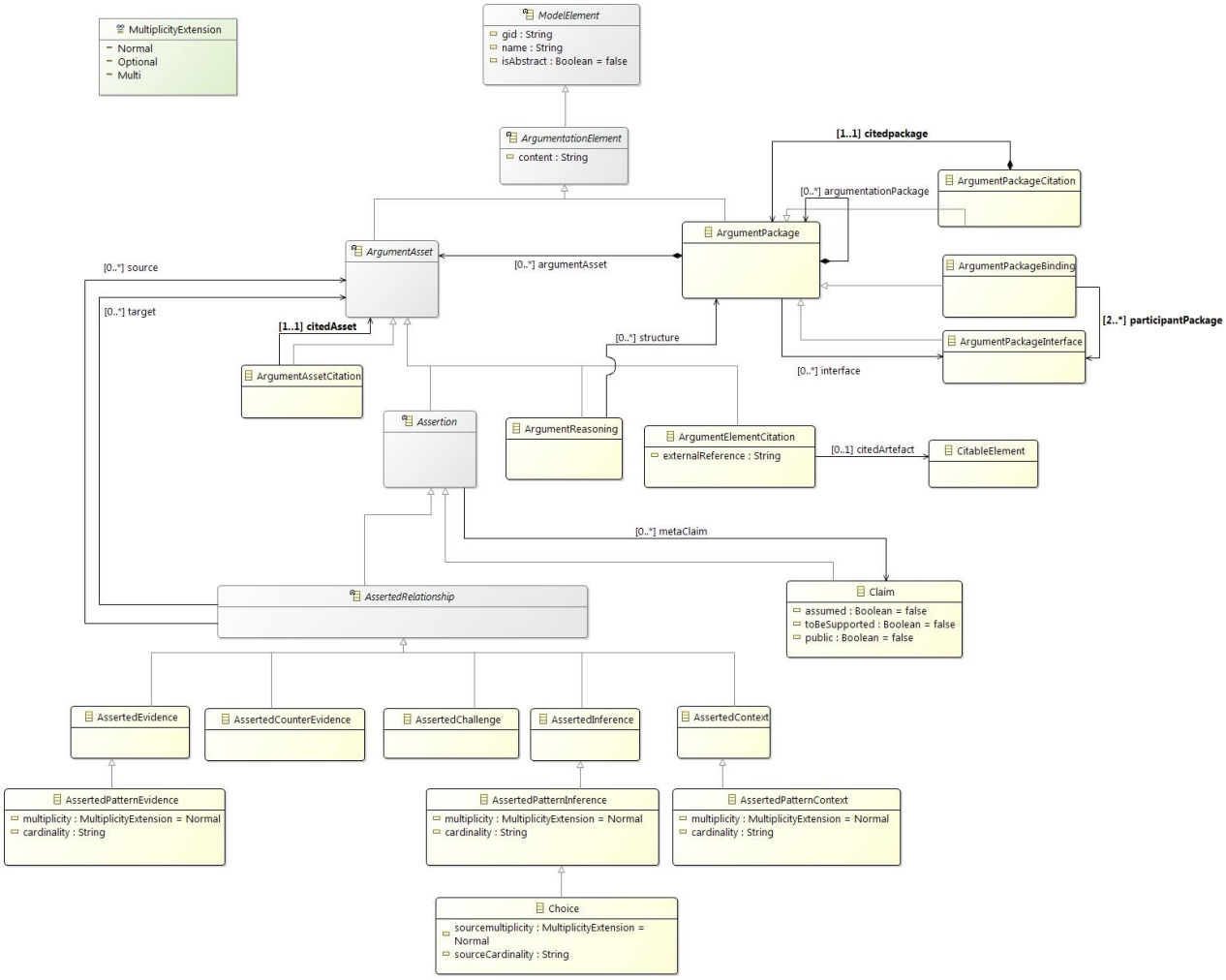


Figure 12. Conceptual Assurance Case Metamodel diagram

This section only describes the classes that are not part of SACM, either because the class is a link with other CACM models or because it is an extension. Please refer to SACM v2 for the rest of the classes description [10].

2.3.2.1 AssertedPatternEvidence

The AssertedPatternEvidence association class records in an argumentation pattern the declaration that one or more items of Evidence (cited by InformationItems) are expected to be included. It is important to note that such a declaration is itself an assertion on behalf of the user.

Superclass

AssertedEvidence

Attributes

- cardinality: String
  An attribute used while specifying patterns to record the number of times the inference should be instantiated afterwards.

Associations

- multiplicity: AssertedMultiplicityExtension
  An attribute used while specifying patterns to indicate whether the inference is multiple, optional or one to one.

Semantics

An AssertedPatternEvidence association between some information cited by an InformationElementCitation and a Claim (A – the source evidence cited – and B – the target claim) denotes that the evidence cited by A is said to help establish the truth of Claim B and needs to be instantiated.

Graphical Notation

multiplicity=optional multiplicity=multi

2.3.2.2 AssertedPatternInference

The AssertedPatternInference association class records the inference in an argument pattern that a user declares to exist between one or more Assertion (premises) and another Assertion (conclusion) and needs to be instantiated.

Superclass

AssertedInference

Attributes

- cardinality: String
  An attribute used while specifying patterns to record the number of times the inference should be instantiated afterwards.

Associations

- multiplicity: AssertedMultiplicityExtension
  An attribute used while specifying patterns to indicate whether the inference is multiple, optional or one to one.

Semantics

The core structure of an argument pattern is declared through the inferences that are asserted to exist between Assertions (e.g., Claims).

Graphical Notation

multiplicity=optional multiplicity=multi

2.3.2.3 Choice

This class is a subtype of the AssertedPatternInference Class. It is used to denote possible alternatives in satisfying an inference in a pattern.

Superclass

AssertedInference

Attributes

- sourceCardinality: String
  An attribute used while specifying patterns to record the number of times the inference should be instantiated afterwards.

Associations

- sourceMultiextension: AssertedMultiplicityExtension
  An attribute used while specifying patterns to indicate whether the source of the inference is multiple, optional or one to one.

Semantics

It is used to denote possible alternatives in satisfying an inference. It can represent 1-of-n and m-of-n selection, an annotation indicating the nature of the choice to be made.

Graphical Notation

2.3.2.4 AssertedPatternContext

The AssertedPatternContext association class shall be included in an argument pattern. It is used to declare that the information cited by an InformationElementCitation provides a context for the interpretation and needs to be instantiated.

Superclass

AssertedContext

Attributes

- cardinality: String
  An attribute used while specifying patterns to record the number of times the context should be instantiated afterwards.

Associations

- multiplicity: AssertedMultiplicityExtension
  An attribute used while specifying patterns to indicate whether the context reference is multiple, optional or one to one.

Semantics

Claim and ArgumentReasoning often need contextual information to be cited and instantiated for the scope and definition of the reasoning to be easily interpreted.

Graphical Notation

multiplicity=optional multiplicity=multi

2.3.2.5 Claim

It is a specialization of an Assertion. It categorizes the content of a proposition with a concern.
Claims are used to record the propositions of any structured Argumentation. Propositions are instances of statements that could be true or false but cannot be true and false simultaneously.

Superclass

Assertion

Attributes

- assumed: Boolean
  An attribute recording whether the claim being made is declared as being assumed to be true rather than being supported by further reasoning.
- toBeSupported: Boolean
  An attribute recording whether further reasoning has yet to be provided to support the Claim (e.g., further evidence to be cited).
- public: Boolean
  An attribute recording whether the proposition described in the claim is publicly visible to other arguments and this way is able to be referenced in other structures of argumentation.

Semantics

The core of any argument is a series of claims (premises) that are asserted to provide sufficient reasoning to support a (higher-level) claim (i.e., a conclusion).

A Claim that is intentionally declared without any supporting evidence or argumentation can be declared as being assumed to be true. It is an assumption. However, it should be noted that a Claim that is not ‘assumed’ (i.e., assumed = false) is not being declared as false.

A Claim that is intentionally declared as requiring further evidence or argumentation can be denoted by setting toBeSupported to be true.

Claims are related with ArgumentElementCitation through the AssertedEvidence relationship. Claims are also related to other claims in a decomposition structure. The AssertedInference relationship is also used to refer to such relationships.

Graphical Notation

assumed=false
toBeSupported=false

assumed=true
toBeSupported=false

assumed=false
toBeSupported=true

assumed=false
toBeSupported=false
public=true

2.3.2.6 CitableElement

It is part of the 2.4.3.2.

2.4 Evidence Management Metamodels

2.4.1 Scope and Purpose

In general terms, the metamodels for evidence management aim at supporting the specification of the information about a project's safety (or assurance) evidence: the artefacts that contribute to developing confidence in the safe operation of a system and that can be used to show compliance with a safety (or assurance) standard. For CPS assurance and certification, an explicit differentiation between what is planned to do in a project and what has actually been done is necessary. The former is specified with the Compliance Management Metamodel, and the latter with the Evidence Management Metamodels.

The metamodels are mostly based on the OPENCOSS metamodels [13] and SACM [10], also taking into account some aspects from DAF [14], SPEM [11], and UMA [12]. The metamodels must allow a user to specify evidence information regardless of the use of other AMASS aspects, such as compliance management and argumentation aspects.

The Evidence Management Metamodels consist of three metamodels: Traceability Metamodel, Managed Artefact Metamodel, and Executed Process Metamodel.

The following subsections present each Evidence Management metamodel.

2.4.2 Conceptual Traceability Metamodel

Traceability can roughly be defined as the existence of a relationship between two artefacts managed in a system's lifecycle. This metamodel (see Figure 13) aims at supporting the specification of the main information related to such relationships, which includes information for change impact analysis.

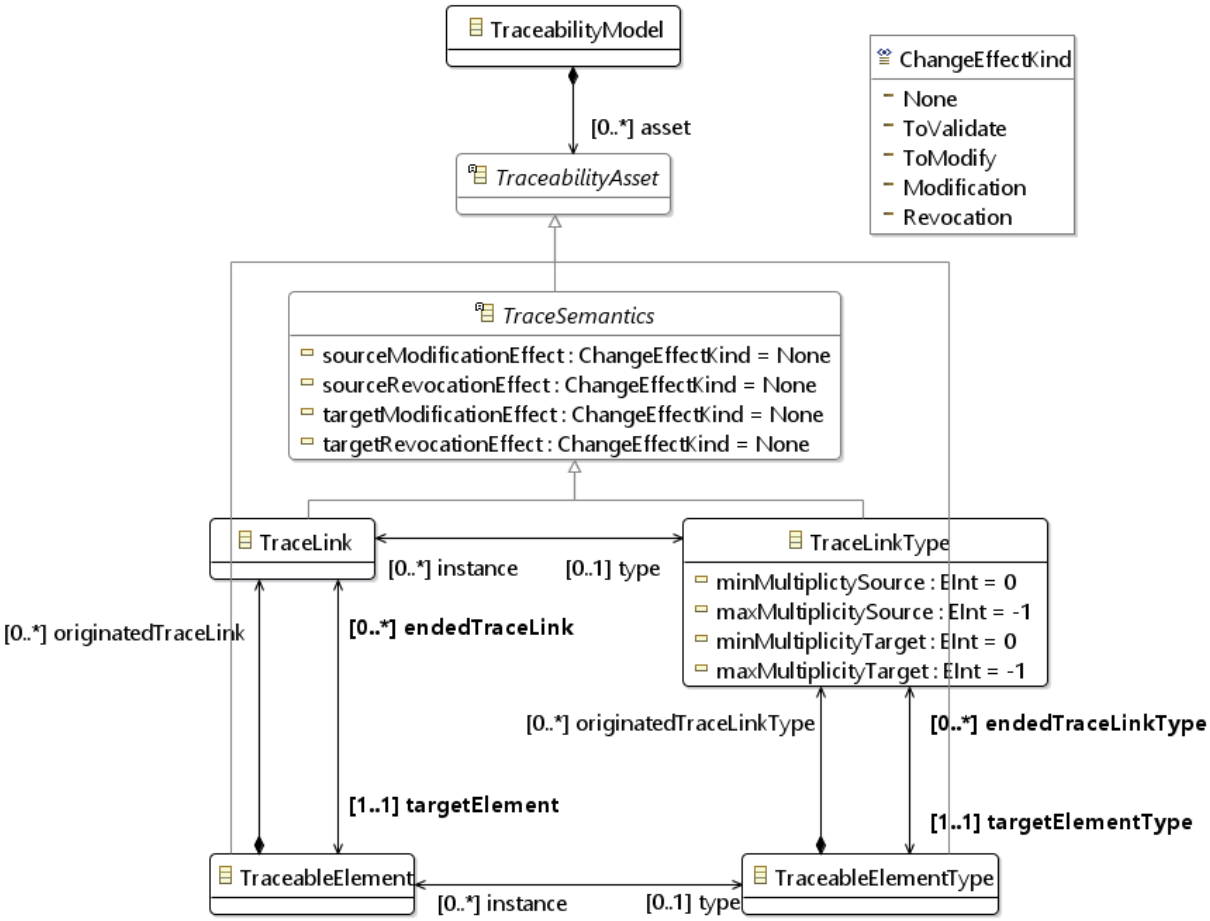


Figure 13. Traceability Metamodel

2.4.2.1 TraceabilityModel

This class corresponds to a model of traceability for a given assurance project.

Superclass

DescribableElement

Associations

- asset: TraceabilityAsset [0..*]
  The TraceabilityAssets of a TraceabilityModel.

Semantics

A TraceabilityModel represents traceability-specific information of an assurance project. Such information corresponds to artefacts (TraceableElement) and relationships (TraceLink) between them: a low-level requirement that refines a high-level one, a test case to validate a low-level requirement, a piece of source code that implements some design block, etc. The artefacts and the relationships can also be of some specific type (TraceableElementType and TraceLinkType, respectively).

2.4.2.2 TraceabilityAsset (abstract)

This class abstracts all the traceability elements of which a TraceabilityModel consists.

Semantics

All the elements of a TraceabilityModel have in common that they belong to the model. This common property is represented in the Traceability Metamodel by means of TraceabilityAsset.

2.4.2.3 TraceSemantics (abstract)

This class abstracts the main link information that a TraceabilityModel can contain, focusing on the semantics of the links.

Superclass
TraceabilityAsset
DescribableElement

Attributes

- sourceModificationEffect: ChangeEffectKind
  The effect that the modification of the source has on the target of a link.
- sourceRevocationEffect: ChangeEffectKind
  The effect that the revocation of the source has on the target of a link.
- targetModificationEffect: ChangeEffectKind
  The effect that the modification of the target has on the source of a link.
- targetRevocationEffect: ChangeEffectKind
  The effect that the revocation of the target has on the source of a link.

Semantics
The links (TraceLink) of a TraceabilityModel and their types (TraceLinkType) share certain characteristics. Such characteristics are abstracted by means of TraceSemantics, and currently correspond to change impact analysis-related information. This information can be used later to: (1) specify how all the instances of a TraceLinkType are expected initially to be analysed for change impact, or; (2) specify how a specific TraceLink has to be analysed for change impact.

2.4.2.4 TraceLinkType

This class represents types of relationships in a TraceabilityModel, i.e. it abstracts relationships that have the same or similar structure (syntax) and/or purpose (semantics).

Superclass
TraceSemantics

Attributes

- maxMultiplicitySource: Int
  The maximum number of times that a TraceableElement that materialises the source of a TraceLinkType can be used as the source of TraceLinks that materialise TraceLinkType.
- minMultiplicitySource: Int
  The minimum number of times that a TraceableElement that materialises the source of a TraceLinkType can be used as the source of TraceLinks that materialise TraceLinkType.
- maxMultiplicityTarget: Int
  The maximum number of times that a TraceableElement that materialises the target of a TraceLinkType can be used as the target of TraceLinks that materialise TraceLinkType.
- minMultiplicityTarget: Int
  The minimum number of times that a TraceableElement that materialises the target of a TraceLinkType can be used as the target of TraceLinks that materialise TraceLinkType.

Associations

- instance: TraceLink [0..*]
  The TraceLinks that materialise a TraceLinkType.
- targetElementType: TraceableElementType [1..1]
  The TraceableElementType that can be the target of a TraceLinkType.

Semantics
In a TraceabiltiyModel, some relationships (TraceLink) can have the same semantics. For example, the semantics of a relationship between a test case and a requirement can mean that the test case *validates* the requirement. Since this semantics could be shared with other relationships, a TraceLinkType could be specified for all the relationships, i.e. for all the relationships between test cases and requirements representing validation. Multiplicity constraints could also be specified. For the example, it could be specified that all the test cases must validate at least one requirement and that all the requirements must be validated by at least one test case. Such constraints are typically indicated in assurance standards. A company can also specialise the constraints or define its own constraints for the types of relationships between artefacts of an assurance project.

2.4.2.5 TraceLink

This class represents the existence of a concrete relationship between two traceable elements.

Superclass
TraceSemantics

Associations
- type: TraceLinkType [0..1]
  The TraceLinkType that a TraceLink materialises.
- targetElement: TraceableElement [1..1]
  The TraceableElement that is the target of the TraceLink.

Semantics
The specific relationships between artefacts (TraceableElement) are represented by means of TraceLink in a TraceabilityModel, e.g. the relationship between a given test case to confirm that a system informs a driver of road conditions and a requirement such as “The system shall indicate when the road is icy”.

2.4.2.6 TraceableElement

This class represents units of data for which relationships can be determined with other units.

Superclass
TraceabilityAsset

Associations
- type: TraceableElementType [0..1]
  The type of a TraceableElement.
- originatedTraceLink: TraceLink [0..*]
  The TraceLinks for which a TraceableElement is the source.
- endedTraceLink: TraceLink [0..*]
  The TraceLinks for which a TraceableElement is the target.

Semantics
The specific artefacts (or elements in general) whose relationships are recorded in a TraceabilityModel are represented by means of TraceableElement. For example, a TraceableElement can correspond to a concrete requirement such as “The system shall indicate when the road is icy”.

2.4.2.7 TraceableElementType

This class represents types of TraceableElements in a TraceabilityModel, i.e. it abstracts TraceableElements that have the same or similar structure (syntax) and/or purpose (semantics).

Superclass
TraceabilityAsset

Associations
- originatedTraceLinkType: TraceLinkType [0..*]

The TraceLinkTypes for which a TraceableElementType is the source.

- endedTraceLinkType: TraceLinkType [0..*]
  The TraceLinkTypes for which a TraceableElementType is the target.

Semantics

The TraceableElements of a TracebilityModel can share characteristics and thus be classified: all the low-level requirements, all the test cases, all the design blocks, etc. Such classification is specified in a TraceabilityModel by means of TraceableElementType. The classification would further support certain usages of the traceability information, such as the derivation of traceability matrices for a pair of TraceabilityElementTypes and the verification of the constraints specified in a TraceLinkType for the TraceElements that correspond to instances of the TraceElementTypes that the TraceLinkType relates.

2.4.2.8 ChangeEffectKind (enumeration)

This enumeration corresponds to the possible effects that a change in some TraceableElement can have in a related TraceableElement.

Literals

- None
  A change has no effect
- ToValidate
  A change requires a validation
- ToModify
  A change requires a modification
- Modification
  A change causes a modification
- Revocation
  A change causes a revocation

2.4.3 Conceptual Managed Artefact Metamodel

This metamodel (see Figure 14) corresponds to an adaptation of the OPENCOSS Artefact Metamodel [13]. The ‘Artefact’ has been renamed a ‘ManagedArtefact’, to clearly differentiate the terminology with UMA [12]. ManagedArtefactPackage has been added to support ManagedArtefact grouping, in line with the most recent work on SACM [10].

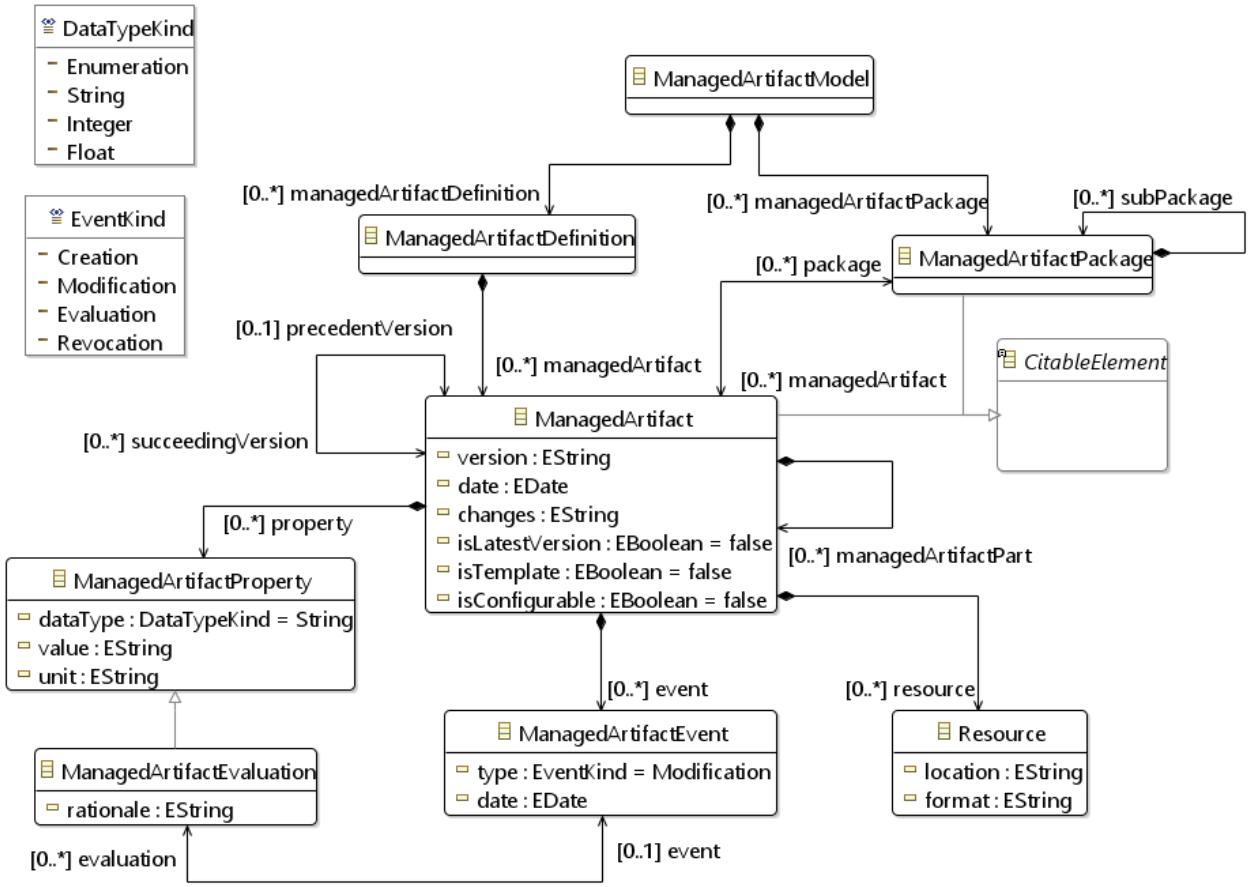


Figure 14. Managed Artifact Metamodel

Managed artifacts are individual concrete units of data managed in an assurance project. The Managed Artifact Metamodel defines the metadata about artefacts which should be captured. In most cases, the classes of metadata and relationships established in this Metamodel can be used to support reasoning about the use of managed artifacts as evidence of compliance with standards or in support of an assurance argument. The Managed Artifact Metamodel links directly to the Executed Process and Argumentation Metamodels.

Managed artifacts are the main evidence information elements of an assurance project. In addition to the information about a managed artefact itself (e.g. its version), other information types can be needed about a managed artifact for CPS assurance and certification, such as its lifecycle or the evaluations performed for it.

2.4.3.1 ManagedArtifactModel

This class corresponds to a model of the specific artefacts managed in a given assurance project.

Superclass

DescribableElement

Associations

- managedArtifactDefinition: ManagedArtifactDefinition [0..*]
  The ManagedArtifactDefinitions of the ManagedArtifactModel.
- managedArtifactPackage: ManagedArtifactPackage [0..*]
  The ManagedArtifactPackages of the ManagedArtifactModel.

Semantics
All the information about the artefacts managed in an assurance project (ManagedArtifact) are represented by means of ManagedArtifactModels, to which the information belongs. The information includes several information types: the versions of the artefacts, their lifecycle, their properties, etc.

2.4.3.2 CitableElement (abstract)

This class corresponds to the information about the specific artefacts managed in a given assurance project that could be cited in other sources of assurance information, e.g. an assurance case.

Semantics
Certain information about the artefacts managed in an assurance project can be cited in other source of assurance information. A classic example is the citation of an artefact in an assurance case so that the artefact is used as evidence of e.g. some claim. CitableElement abstracts the information from a ManagedArtifactModel that can be cited: ManagedArtifact and ManagedArtifactPackage.

2.4.3.3 ManagedArtifactDefinition

This class corresponds to a distinguishable abstract unit of data to manage in an assurance project that depicts the whole lifecycle resulting from the evolution, in different versions, of ManagedArtifacts. A ManagedArtifactDefinition would be specified for e.g. a hazard log. Each requirement of a requirements specification would have its own ManagedArtifactDefinition.

Superclass
DescribableElement

Associations
- managedArtifact: ManagedArtifact [0..*]
  The ManagedArtifacts of the ManagedArtifactDefinition

Semantics
The artefacts managed in an assurance project can evolve during the project. For example, a hazard log can be created at the beginning of a project and it will be updated as safety requirements are specified, verification measures are defined, verification actions are taken, etc. In other words, the hazard log will have different versions during the project. Each individual version is represented in a ManagedArtifactModel by means of ManagedArtifact (see next class description). In an assurance project, it is necessary not only to record the information about the individual versions, but also to keep track in a common place of the different versions, of how the artefact has evolved. This is done with ManagedArtifactDefinition. Intuitively, ManagedArtifactDefinition records all the information of the lifecycle of an artefact of an assurance project, i.e. it represents the lifecycle of the artefact.

2.4.3.4 ManagedArtifact

This class correspond to a distinguishable, concrete, and individual unit of data managed in an assurance project.

Superclass
DescribableElement
CitableElement

Attributes
- version: String
  The version ID of the ManagedArtifact.
- date: Date
  The date of creation of the ManagedArtifact.
- changes: String
  The list of changes describing any update regarding the previous version.

- isLatestVersion: Boolean
  A flag to indicate if the ManagedArtifact version is the latest one.
- isTemplate: Boolean
  A flag to indicate if the ManagedArtifact is a Template to create the actual ManagedArtifact to be used in an assurance project.
- isConfigurable: Boolean
  A flag to indicate if the ManagedArtifact can be configured for specific usage contexts or situations.

Associations

- package: ManagedArtifactPackage [0..*]
  The set of ManagedArtifactPackages that refer to a ManagedArtifact.
- property: ManagedArtifactProperty [0..*]
  The set of ManagedArtifactProperties of a ManagedArtifact.
- event: ManagedArtifactEvent [0..*]
  The ManagedArtifactEvents of which the lifecycle of a ManagedArtifact consists.
- resource: Resource [0..*]
  The Resources that represent the tangible objects of a ManagedArtifact, e.g. the set of architectural model files of an Architecture Design document.
- precedentVersion: ManagedArtifact [0..1]
  A pointer to the precedent version of a ManagedArtifact.
- succeedingVersion: ManagedArtifact [0..*]
  A pointer to the succeeding version of a ManagedArtifact.
- managedArtifactPart: ManagedArtifact [0..*]
  The parts of a ManagedArtifact that can represent document sections or any element composing the whole ManagedArtifact.

Semantics

In contrast to ManagedArtifactDefinition, which is used to represent the whole lifecycle of an artefact of an assurance project, ManagedArtifact is used to represent the individual versions of an artefact during an assurance project. This is necessary to be able to ascertain the version of an artefact that has been used as evidence of a claim in an assurance case, the version that has been used as input in a given activity, or the version that is related to another artefact, among other usages.

2.4.3.5 ManagedArtifactPackage

This class corresponds to a group of ManagedArtifacts that can be referred together, e.g. in an assurance case.

Superclass

DescribableElement
CitableElement

Associations

- subPackage: ManagedArtifactPackage [0..*]
  ManagedArtifactPackages of which a ManagedArtifactPackage consists
- managedArtifact: ManagedArtifact [0..*]
  The ManagedArtifacts grouped in a ManagedArtifactPackage

Semantics

The ManagedArtifacts of a ManagedArtifactModel are usually referred to individually, e.g. in an assurance case. However, sometimes it is necessary to refer to them as a group, e.g. to all the ManagedArtifacts that contain software V&V results or that correspond to process evidence. Such group is an abstract representation of the ManagedArtifacts, as the group does not exist in reality. For the examples, there will be different concrete individual artefacts that are software V&V results or process evidence.

ManagedArtifactPackage is used to specify such groups and also refer to them in other sources of assurance information as a CitableElement.

2.4.3.6 ManagedArtifactProperty

This class corresponds to a characteristic of a ManagedArtifact, generally depicted with an attribute (name) and a value.

Superclass
DescribableElement

Attributes

- dataType: DataTypeKind
  The type of the data used to represent the values of a ManagedArtifactProperty.
- value: String
  The value of a ManagedArtifactProperty.
- unit: String
  The measurement unit corresponding to ManagedArtifactProperty values.

Semantics
ManagedArtifacts can have characteristics that must be recorded for assurance purposes. For example, it is necessary to know whether a given test case is passed or not. Such characteristics are represented in a ManageArtifactModel by means of ManagedArtefactProperty. ManagedArtefactProperties correspond to objective characteristics. For judgement-based (or subjective) characteristics, ManagedArtifactEvaluation must be used.

2.4.3.7 ManagedArtifactEvaluation

This class corresponds to the specification of the result of making some judgement regarding a ManagedArtifact.

Superclass
ManagedArtifactProperty

Attributes

- rationale: String
  The justification of the value of a ManagedArtifactEvaluation

Associations

- event: ManagedArtifactEvent [0..1]
  The ManagedArtifactEvent at which a ManagedArtifactEvaluation is made.

Semantics
In addition to ManagedArtifactProperties, it can also be necessary to record judgement-based characteristics of ManagedArtifacts for an assurance project, typically about their quality (completeness, accuracy, consistency, etc.). It is important to record the rationale behind a ManagedArtifactEvaluation to understand the judgement made and how it has led to the evaluation result.

2.4.3.8 ManagedArtifactEvent

This class corresponds to relevant happenings in the lifecycle of a ManagedArtifact. This serves to maintain a history log for ManagedArtifacts.

Superclass
DescribableElement

Attributes

- type: EventKind
  The type of happening of a ManagedArtifactEvent.

- date: Date
  The date (and time) when a ManagedArtifactEvent occurred.

Associations

- evaluation: ManagedArtifactEvaluation [0..*]
  The ManagedArtifactEvaluations that result from a ManagedArtifactEvent.

Semantics

ManagedArtifacts change during an assurance project: someone creates it, someone makes some modification, someone fixes some problems, etc. ManagedArtifactEvent is used to represent the happening that corresponds to these changes. ManagedArtifactEvents can be consulted to know how ManagedArtifact has evolved and to develop confidence in its adequate management.

2.4.3.9 Resource

This class corresponds to the tangible objects representing a ManagedArtifact.

Superclass

DescribableElement

Attributes

- location: String
  The path or URL specifying the location of the Resource.
- format: String
  The format of the resource, e.g. MS Word.

Semantics

ManagedArtifacts are located and accessible somewhere, usually in the form of some electronic file: a Word file, an Excel file, a file created with some modelling tool, etc. Such information is specified by means of Resource.

2.4.3.10 DataTypeKind (enumeration)

This enumeration corresponds to types of values for ManagedArtifactProperties.

Literals

- Enumeration
  The value space characterized for a list of qualitative values.
- String
  The value space characterized by a string.
- Integer
  A value space characterized by integer numbers.
- Float
  A value space characterized by real numbers.

2.4.3.11 EventKind (enumeration)

This enumeration corresponds to types of events that can occur in the lifecycle of a ManagedArtifact.

Literals

- Creation
  When a ManagedArtifact is brought into existence.
- Modification
  When a change is made in some characteristic of a ManagedArtifact.
- Evaluation
  When a ManagedArtifact is evaluated.
- Revocation

When a ManagedArtifact is revoked from an assurance project.

2.4.4 Conceptual Executed Process Metamodel

The artefacts that are used as evidence in an assurance project are employed in and are the result of processes during a product's lifecycle. The information about such processes is specified with the Executed Process Metamodel (Figure 15).

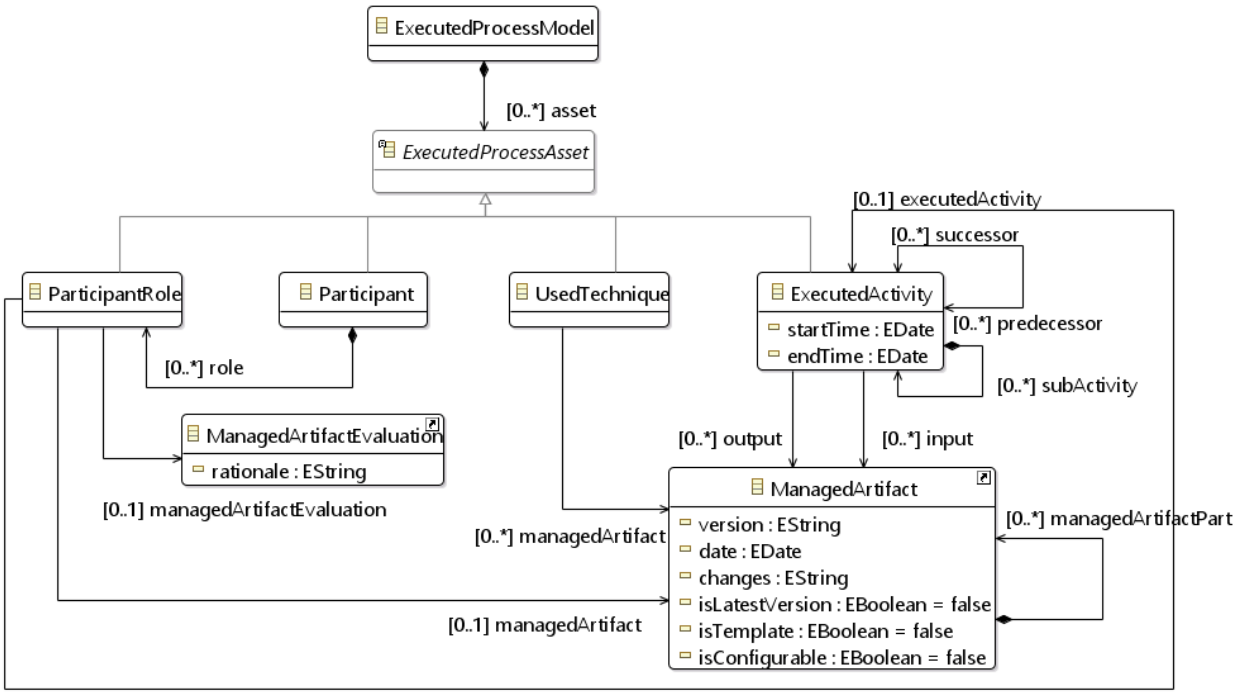


Figure 15. Executed Process Metamodel

2.4.4.1 ExecutedProcessModel

This class corresponds to a model of the process executed in a given assurance project in relation to managed artifacts.

Superclass
DescribableElement

Associations

- asset: ExcutedProcessAsset [0..*]
  The assets of the ExecutedProcessModel.

Semantics
An ExecutedProcessModel contains information specific to the process performed, as part of an assurance project, to manage the artefacts of the project (ManagedArtifacts). Such information allows a person to e.g. know when a ManagedArtifact has been the input or output of some activity, how the ManagedArtifact has been created, and what people have been involved in its lifecycle.

2.4.4.2 ExecutedProcessAsset

This class corresponds to the assets of which an ExecutedProcessModel consists.

Superclass
DescribableElement

Semantics
All the main elements of an ExecutedProcessModel have a characteristic in common: they belong to the model. ExecutedProcessAsset represents this common characteristic.

2.4.4.3 ExecutedActivity

This class corresponds to a unit of work performed in a product lifecycle.

Superclass
ExecutedProcessAsset

Attributes
- startTime: Date
  When an ExecutedActivity starts.
- endTime: Date
  When an ExecutedActivity ends.

Associations
- input: ManagedArtifact [0..*]
  The ManagedArtifacts used in an ExecutedActivity.
- output: ManagedArtifact [0..*]
  The ManagedArtifacts produced or changed in an ExecutedActivity.
- subActivity: ExecutedActivity [0..*]
  ExecutedActivities of which an ExecutedActivity consists.
- successor: ExecutedActivity [0..*]
  The ExecutedActivities that are performed after an ExecutedActivity.
- predecessor: ExecutedActivity [0..*]
  The ExecutedActivities that are performed before an ExecutedActivity.

Semantics
The ManagedArtefacts used in an assurance project are the result of and are managed via the execution of processes, which consist of activities: specification of requirements, design of the system, integration of system components, etc. ExecutedActivities are used in an ExecutedProcessModel to represent these activities.

2.4.4.4 UsedTechnique

A specific way to create a ManagedArtifact.

Superclass
ExecutedProcessAsset

Associations
- managedArtifact: ManageArtifact [0..*]
  The ManageArtifacts that are created with an UsedTechnique.

Semantics
ManagedArtifacts are created, or managed from a more general perspective, via some technique (methods, approaches, languages…) whose use results in specific characteristics for the ManagedArtifacts. For example, the use of UML for designing a system results in a design specification with a set of UML diagrams that could represent static and dynamic internal aspects of the system. UsedTechniques of an ExecutedProcessModel support the specification of this kind of information.

2.4.4.5 Participant

This class corresponds to the concrete parties involved in a product lifecycle, e.g. a person.

Superclass

ExecutedProcessAsset

Associations

- role: ParticipantRole [0..*]
  The roles that a Participant plays in a product lifecycle.

Semantics

Different parties can participate in an assurance case effort, such as specific people, organizations, and tools. These parties are represented by means of Participant.

2.4.4.6 ParticipantRole

This class enables a Participant to specify its roles.

Superclass

ExecutedProcessAsset

Associations

- executedActivity: ExecutedActivity [0..1]
  An ExectuedActivitiy in which a participant is involved.
- managedArtifactEvaluation: ManagedArtifactEvaluation [0..1]
  A ManagedArtifactEvaluation in which a participant is involved.
- managedArtifact: ManageArtifact [0..1]
  A ManagedArtifact in whose management a participant is involved.

Semantics

The information about the roles and functions that a Participant plays in an assurance project is specified by means of ParticipantRole. Examples of roles and functions include the owner of a ManagedArtifact, the executor of an Activity, and possible relationships between Participants (e.g., supervisor).

2.4.4.7 ManagedArtifact

See definition in 2.4.3.4.

2.4.4.8 ManagedArtifactEvaluation

See definition in 2.4.3.7.

2.5 Compliance Management Metamodel

2.5.1 Scope and Purpose

The compliance metamodel contains the necessary concepts to model:

a) **Assurance project definition**. It is used to define the assets produced during the development, assessment and justification of a safety-critical system.

b) **Process Definition**. It is used to model general reference processes (e.g., Waterfall Process, Agile Process, the so-called V-model), or company-specific processes (e.g., the Thales process to develop safety-critical systems).

c) **Standard Definition**. It is used to model standards (IEC 61508 [15], ISO 26262 [16], DO-178C [17], EN 50126 [18], and the like) and any regulations (either as additional Requirements or model elements in a given model representing a standard or a new reference standard). For the implementation another metamodel is added, the Baseline Metamodel, to capture what is planned to be done or to be complied with a defined standard, in a concrete assurance project.

d) Vocabulary Definition. It is used to provide a Thesaurus-type vocabulary, which defines and records key concepts relevant to safety assurance within the target domains and the relationships between them.

e) **Mapping Definition**. It is used to capture the nature of the vertical and horizontal mappings between the different levels of model in the AMASS Framework and between the concepts and vocabulary used in these models. There are two types of mapping: one mapping type, called Equivalence Mapping, maps process models with models of standards; the other mapping maps process-models and project specific models.

Figure 1 at the beginning of the document, describes the CACM approach, where these three sub-metamodels are presented.

The **Process Definition metamodel** is based on the UMA (Unified Method Architecture) metamodel. AMASS has adopted UMA in order to fully reuse the EPF (Eclipse Process Framework) Composer tool. The EPF/UMA implementation is aligned to the OMG SPEM 2.0 [3].

The conceptual framework of SPEM 2.0 considers two views, the Method content and the Process packages (see Figure 16). The goal of the Method Content package is to set up a knowledge base of intellectual capital for software development that would allow them to manage and deploy their content using a standardized format. Elements that are defined in this package are tasks, roles, tools and support material like white papers, principles or best practices. In contrast, the Process package focus on supporting the systematic development, management, and growth of development processes. This is the package in which development processes are actually defined using elements that point to elements of the Method Content package. In order to define a process, tasks are organized in activities, phases and iterations.

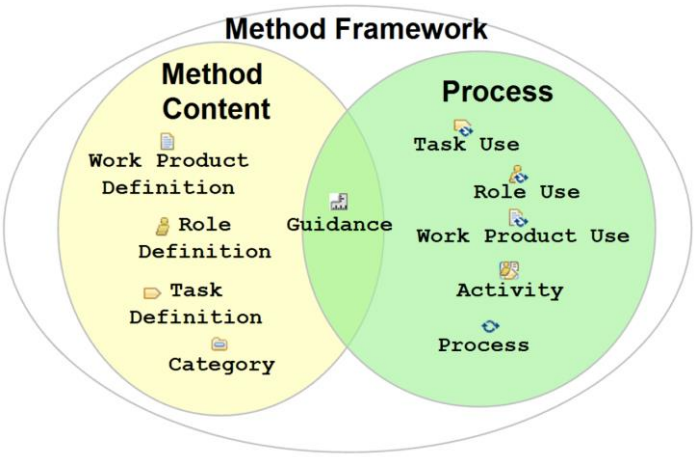


Figure 16. Method Content versus Process in the SPEM 2.0 standard, taken from [11]

The **Standard Definition metamodel** provides the structure to model different kinds of process-based or product-based industrial standards. Each industry standard has its own requirements to fulfil, and such requirements can vary among standards. For example, DO-178C lists objectives (requirements) for the different software development processes, and some objectives are not fully addressed in EN 50128. The above need can only be met if the standard's requirements are recorded. These requirements correspond to the process and product intent: what properties are assured and thus why the different activities and artefacts are necessary. For example, the DO-178C Software Requirements Data must include the performance criteria, timing requirements and constraints, and memory size constraints so that the artefact fulfils its intent (i.e., to show that such characteristics have been considered and specified).

The standard definition metamodel also supports modelling of criticality levels or levels of integrity. Under various names but addressing the same aim, they constitute a fundamental basis of dependability standards. They are called Safety Integrity Levels (SILs) in IEC 61508 and railway, Automotive Safety Integrity Level

(ASIL) in the automotive domain, and Development Assurance Level (DAL) in avionics. A given level corresponds to one of several levels to determine the item or element necessary requirements of the functional safety standard, and the dependability measures to apply for avoiding an unreasonable residual risk. The requirements to comply with increase as the criticality level does. This is the case when dealing with specific techniques or methods for a certain design phase. Depending on the criticality level to address, either complementary techniques or more exhaustive ones are needed to comply with the standard.

The **Mapping Definition metamodel** is the basis to manage compliance. This metamodel is based on the CCL metamodel. There are two kinds of mappings:

- *Equivalence mapping between processes/standards*. It helps to model the equivalence between standards and process or between different standards. This can be done by means of maps that indicate the extent to which the criteria of the standards/process are equal (e.g., between the company-process and what a given industrial standard recommends). Based on these mappings, we can assess the similarity and differences of the standards.

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

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

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

  Three elements play a role in equivalence mapping: artefacts, activities and requirements. Pragmatically, any of these elements can be compared for equivalence modelling.

- *Compliance mapping*. These mappings specify how the information of a project (i.e., the executed activities, produced artefacts and arguments) complies with its project plan. As equivalence maps, compliance maps can be full, partial, or no map. This metamodel is based on the CCL metamodel.

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

2.5.2 Conceptual Assurance Project Definition

This Metamodel defines the assets produced during the development, assessment and justification of a safety-critical system, including those associated with justifying the safety of the system and – in the regulated domains – seeking regulatory approval for its entry into service. The Metamodel describes both tangible assets – artefacts such as documents, plans and so forth -, and intangible ones, such as personnel,

techniques and the like. The Assurance Project Metamodel links directly to the System Definition, Process Definition, Process execution, Artefact and Argumentation Metamodels.

The class diagram for the Assurance Project Definition Metamodel is presented in the figure below.

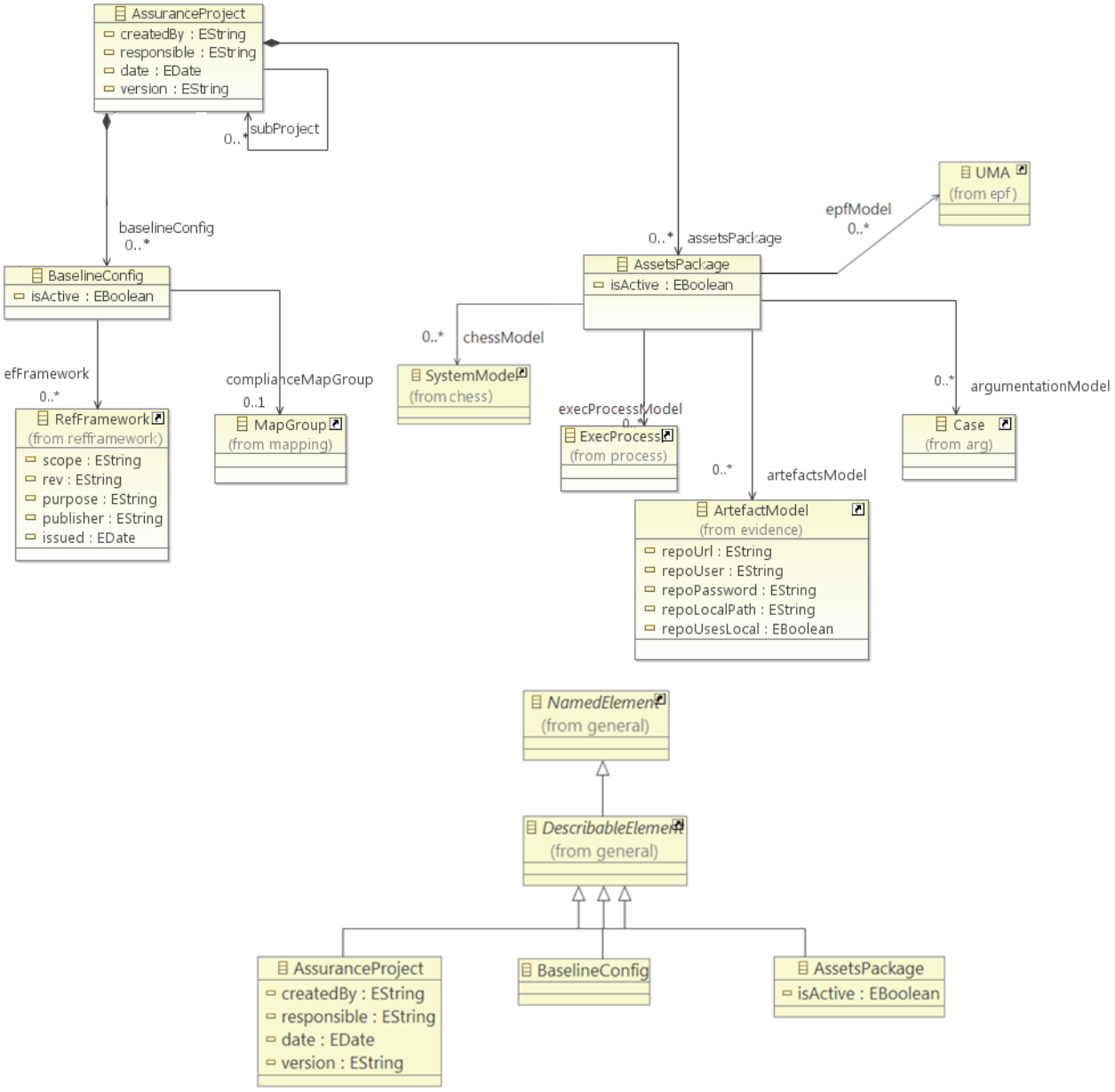


Figure 17. Assurance Project Metamodel

2.5.2.1 AssuranceProject

This class corresponds to an individual or collaborative assurance project that aims to manage the lifecycle of a critical system and to assure its safety or other properties of the system.

Attributes

- createdBy: *String*
  The name of the person who created the Assurance Project.
- responsible: *String*
  The name of the person who is responsible for the Assurance Project.
- date: *Date*

The date of creation of the Assurance Project.

- version: *String*
  The version reference of the Assurance Project.

Relationships

- chessModel: *SystemModel* [0..*]
  The System Model points to all the Chess Model describing the System Architecture related to the Assurance Project.
- baselineConfig: *BaselineConfig* [0..*]
  The Baseline Configuration points to a Baseline model (RefFramework root element in the Baseline Metamodel) and to a Compliance Map Group. A Baseline model represents what is planned to do or to comply with, in a specific assurance project.
- assetsPackage: *AssetsPackage* [0..*]
  This is a pointer to project-specific Artefacts models, Argumentation models, and Process models. These three kinds of models represent what has been done in a specific assurance project.
- subProject: *AssuranceProject* [0..*]
  A reference to a sub-project if the assurance project has been broken down into sub-projects.

Semantics

An Assurance Project models an individual or collaborative assurance project that aims to manage the lifecycle of a critical system and to assure its safety or other properties of the system. One Assurance Project can have multiple Baseline Configurations, Permissions Configurations and Assurance Assets Package, but only one is active at a given time.

2.5.2.2 BaselineConfig

This class corresponds to a pointer to a Baseline model (RefFramework root element in the Baseline Metamodel) and to a Compliance Map Group. A Baseline model represents what is planned to do or to comply with, in a specific assurance project.

Attributes

- isActive: *Boolean*
  It indicates if the Baseline Configuration is active in the context of the Assurance Project.

Relationships

- refFramework: *RefFramework* [0..*]
  The Reference Framework that models what is planned to be done in an assurance project.
- complianceMapGroup: *MapGroup* [0..*]
  The Map Group that is used in the Baseline Configuration to link what has been done (AssetsPackage) with what was planned to be done (BaselineConfig).

Semantics

A Baseline Configuration models a pointer to a Baseline model (RefFramework root element in the Baseline Metamodel) and to a Compliance Map Group. A Baseline model represents what is planned to do or to comply with, in a specific assurance project.

2.5.2.3 AssetPackage

This class corresponds to project-specific Artefacts models, Argumentation models, and Process models. These three kinds of models represent what has been done in a specific assurance project.

Attributes

- isActive: *Boolean*
  It indicates if the Assets Package is active in the context of the Assurance Project.

Relationships

- chessModel: *SystemModel* [0..*]

The System Model points to all the System Architecture Models created in CHESS related an Assurance Project

- execProcessModel: Exec*Process* [0..*]
  The Process Execution Model that specifies the activities and other process-related information executed in an assurance project.
- artefactsModel: *ArtefactModel* [0..*]
  The Artefact Model that collects the set of artefacts produced in an assurance project, as required for compliance purposes.
- argumentationModel: *AssuranceCase* [0..*]
  The Argumentation Model that is the main assurance case for the whole assurance project.
- epfModel: *UMA* [0..*]
  The UMA Model points to all the Process Definition Models created in EPF Composer related to an Assurance Project

Semantics

An Assets Package models project-specific Process Definition Models, System Architecture models, Artefacts models, Argumentation models, and Process models. These five kinds of models represent what has been done in a specific assurance project.

2.5.3 Conceptual Process Definition Metamodel

This metamodel provides an overview of process structure, as shown in Figure 18. The implemented metamodel is called UMA that is described in Section 3.5.3. The meta-classes represent the roles, work products, units of work (e.g., tasks), tool(s), guidance(s) as well as processes.

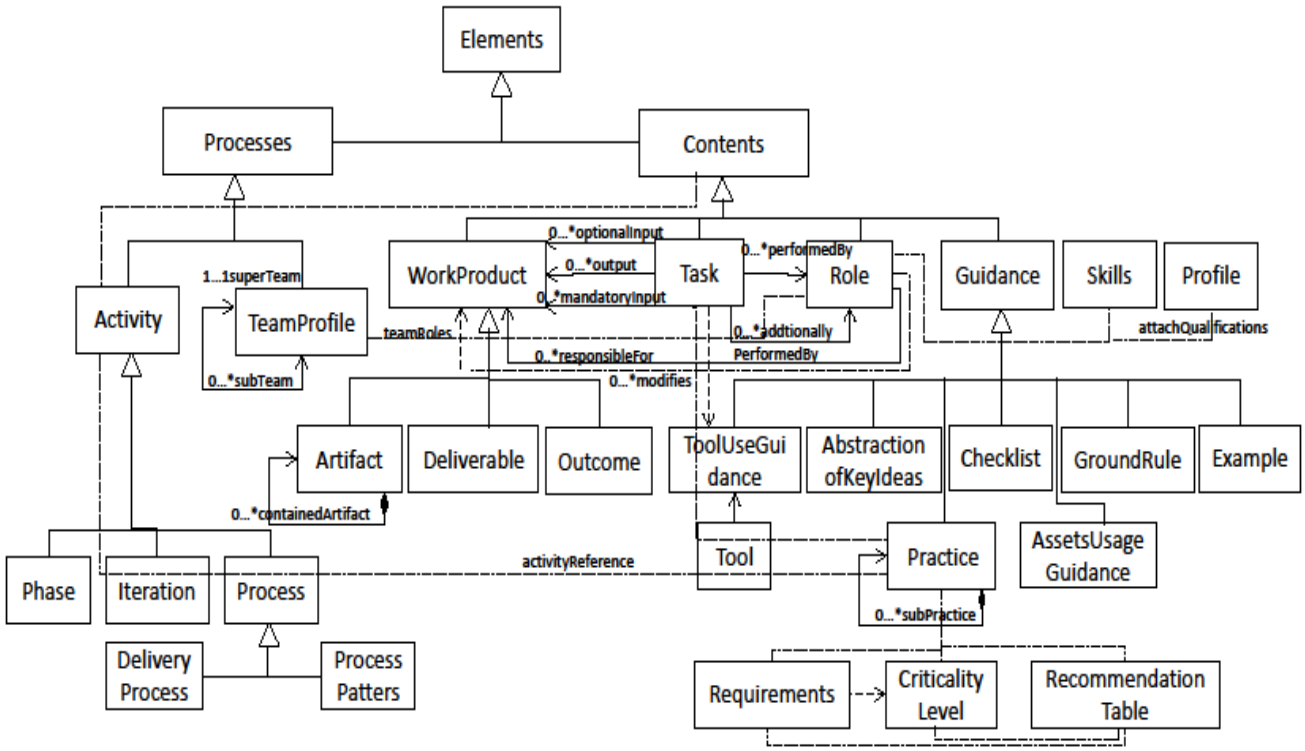


Figure 18. Conceptual Process Definition Metamodel

The core elements can be described as follows:

- Contents - this class corresponds to content elements such as role, task, work product (artifact, outcome and deliverable) and guidance.

- Processes - this class reuses the contents elements and combines them into a structure and sequence of tasks performed by roles and the work products produced over time. The main process element is the activity, used to define process. A process might contain a definition of phases or iterations.
- Process patterns are reusable process elements while Delivery Process describes a complete end-to-end project lifecycle.
- Guidance - checklist, example, ground rule, tool use guidance, abstraction of key ideas, assets usage guidance and practice.
- The guidance type practice linked with process elements to model the requirements and recommendation tables.
- Skills and Profile define the personnel qualifications for a role who is responsible for performing certain tasks and creating certain work products.
- TeamProfile defines teams and roles who will participate in activities.

The integration of Conceptual Process Definition Metamodel with architecture design (CHESSML) and variability management (BVR) has also been considered. In this sense, the process engineering, architecture design and variability management are linked. This, for example, is useful to track the processes and their corresponding architecture design, as well as the configuration of processes and products for individual projects (i.e., their variability management). The interested reader may refer to [25] for the description of BVR metamodel. The CHESSML is defined in Section 2.2. Furthermore, the rule set is defined in a customized reusable asset, which contains the superiority relations and compliance annotation defines to compile the process with the Regorous (Please refer to the D6.3 deliverable [7]).

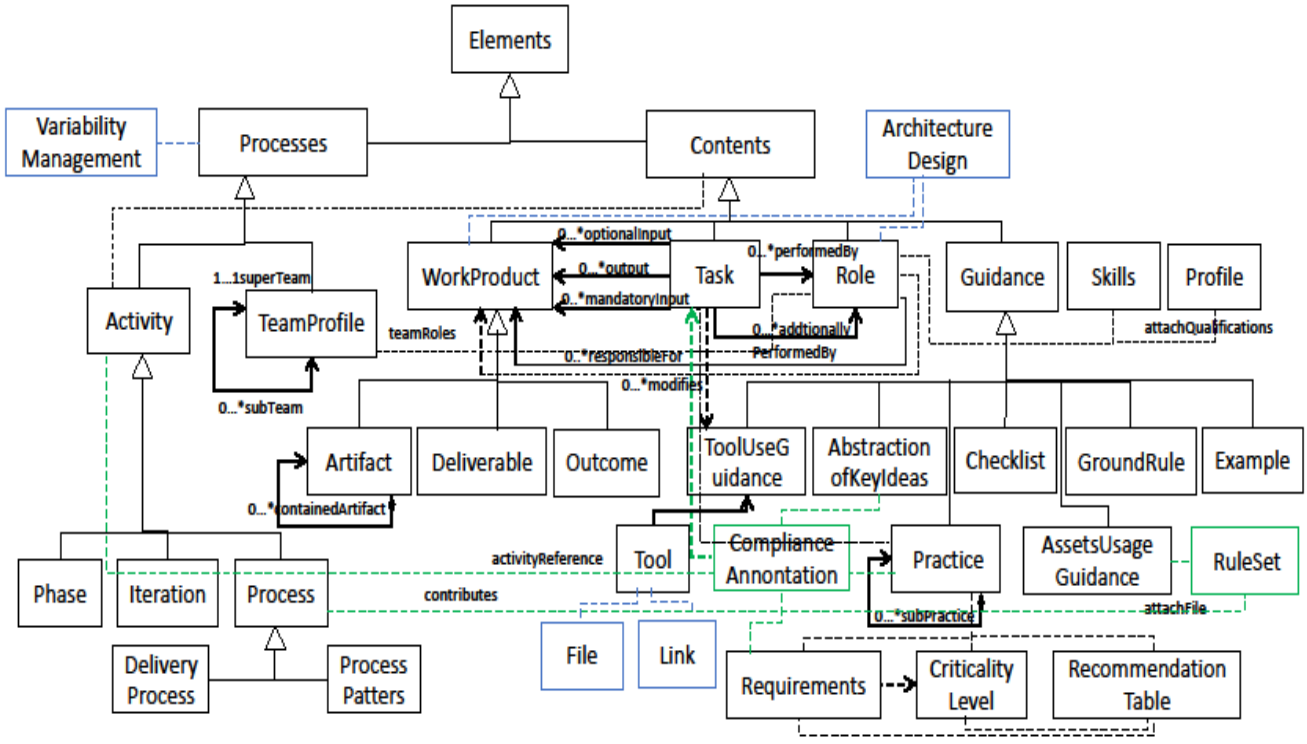


Figure 19. Conceptual Process Definition Metamodel with System Design and Variability Management

2.5.4 Conceptual Standard Definition Metamodel

The Standard Definition Metamodel captures the high-level concepts and relationships required to model a Reference Assurance Framework, a framework against which the safety aspects of a given system are developed and assessed. The concept of a Reference Assurance Framework was referred to as a “Safety Standard”: this concept reflects the fact that a given project does not necessarily develop *directly* to a safety

standard. The Reference Assurance Framework concept encompasses, for example, company standards and best practice documentations (e.g. the Alstom, Thales or Fiat processes to develop safety-critical systems), as well as documents which have the *de facto* status of standards (e.g., IEC 61508, ISO 26262, DO-178C, EN 50126), but are not – legally – such as, for example, the Aerospace Recommended Practice (ARP) documents (e.g. ARP 4754 *Certification Considerations for Highly-Integrated or Complex Aircraft Systems*) [19].

The class diagram for the Standard Definition Metamodel is shown in the figure below. In the following subsections, we define the model elements.

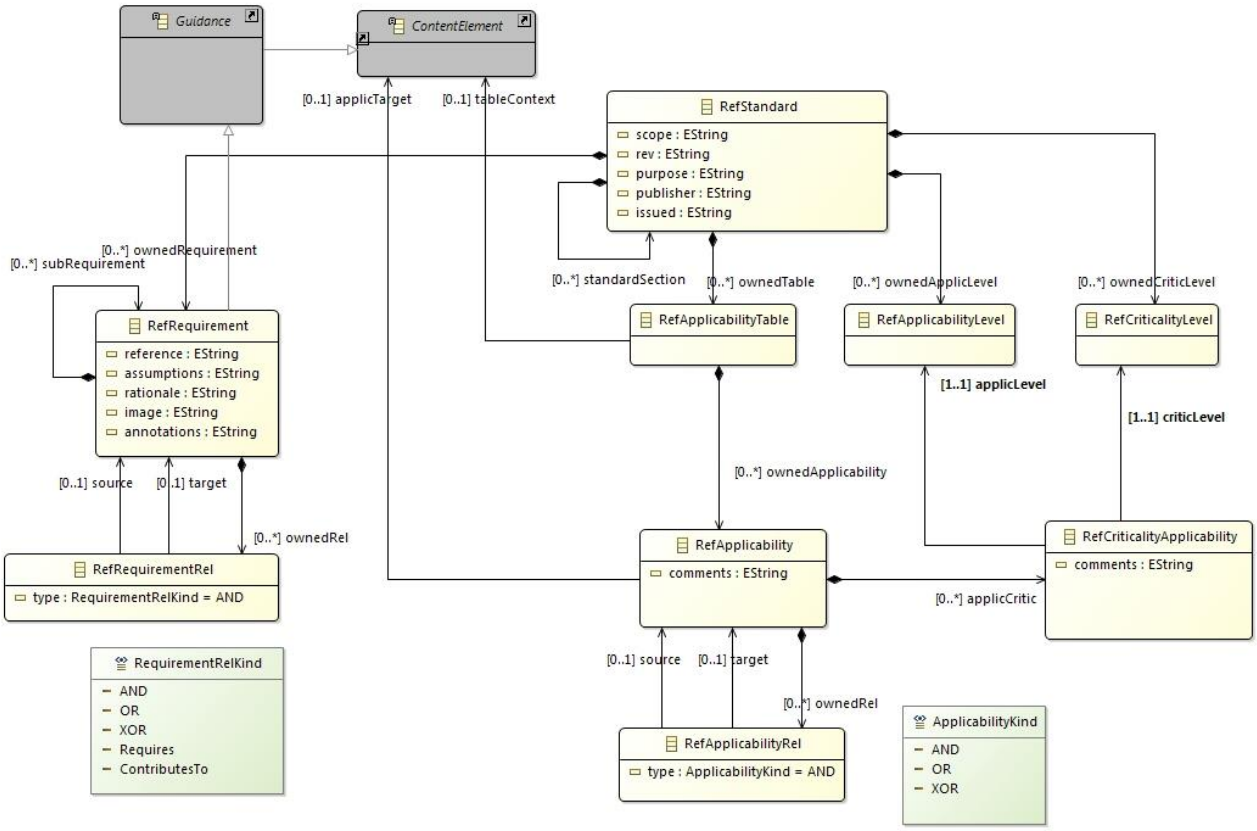


Figure 19. Standard Definition metamodel

2.5.4.1 RefStandard

This class corresponds to a standard to which the lifecycle of a critical system might have to show compliance (for example, *IEC 61508 standard [15]*).

Superclass

- *DescribableElement*

Attributes

- *scope: String*
  The scope of the reference standard.
- *rev: String*
  The revision (version) of the reference standard.
- *purpose: String*
  The purpose of the reference standard.
- *publisher: String*
  The publisher of the reference standard.
- *issued: Date*
  The issue date of the reference standard.

Associations

- ownedRequirement: *RefRequirement* [0..*]
  The (compliance) requirements defined in a reference standard.
- ownedCriticlevel: *RefCriticalityLevel* [0..*]
  The criticality levels defined in a reference standard.
- ownedApplicLevel: *RefApplicabilityLevel* [0..*]
  The applicability levels defined in a reference standard.
- ownedTable: *RefApplicabilityTable* [0..*]
  The applicability table defined in a reference standard.

Semantics

A Reference Standard is the main container to model concepts against which the safety and system engineering aspects of a given system are developed and assessed, for example, safety standards such as IEC 61508 [15], ISO 26262 [16], DO-178C [17], EN 50126 [18], as well as documents which have the de facto status of standards, such as, for example, the Aerospace Recommended Practice (ARP) documents (e.g. ARP 4754 *Certification Considerations for Highly-Integrated or Complex Aircraft Systems [19]*).

2.5.4.2 RefRequirement

This class corresponds to the criteria (e.g., objectives) that a reference standard defines (or prescribes) to comply with it.

Superclass

- *DescribableElement*
- *Guidance*

Attributes

- reference: String
  The reference of the requirement in the reference standard documents.
- assumptions: String
  The statements considered as preconditions to meet the reference requirement.
- rationale: String
  Any rationale to justify the need to meet the reference requirement.
- image: String
  A placeholder for an image capturing the reference requirement description from the reference standard documents.
- annotations: *String*
  Any complementary annotation clarifying the means to meet the requirement.

Associations

- subRequirement: *RefRequirement* [0..*]
  A more fine-grained reference requirement of which this reference requirement is composed.
- ownedRel: *RefRequirementRel* [0..*]
  The reference requirement relationships owned by a reference requirement.

Semantics

The Reference Requirement models the criteria (e.g., objectives) that a reference standard defines (or prescribes) to comply with it.

2.5.4.3 RefRequirementRel

This class corresponds to the existence of a relationship between two requirements.

Attributes

- type: *RequirementRelKind*
  The kind of a requirements relationship.

Associations

- source: *RefRequirement* [1]
  The reference requirement that corresponds to the source of a reference requirement relationship.
- Target: *RefRequirement* [1]
  The reference requirement that corresponds to the target of a reference requirement relationship.

Semantics

A Reference Requirement Relationship models different kinds of relationships between two reference requirements. The semantics of the relationships are defined by the RequirementRelKind enumeration.

2.5.4.4 RefCriticalityLevel

This class corresponds to the categories of criticality that a reference assurance framework defines and that indicate the relative level of risk reduction being provided (e.g., *SIL 1, 2, 3, and 4* for IEC61508).

Superclass

- *DescribableElement*

Semantics

This Reference Criticality Level models the categories of criticality that a reference assurance framework defines and that indicate the relative level of risk reduction that needs to be provided (e.g., *SIL 1, 2, 3, and 4* for IEC61508).

2.5.4.5 RefApplicabilityLevel

This class corresponds to the categories of applicability that a reference standard defines (e.g., a given technique can be *mandated* in EN50128).

Superclass

- *DescribableElement*

Semantics

This Reference Applicability Level models the categories of applicability that a reference standard defines (e.g., a given technique can be *mandated* in EN50128).

2.5.4.6 RefIndependencyLevel

This class corresponds to the kind of categories of applicability related to the independency required to perform an activity or to achieve and compliance objective that a reference standard defines (e.g., the level of independence of the person performing a verification activity *mandated* in DO-178C).

Superclass

- *RefApplicabilitylevel*

Semantics

This Reference Independency Level models the kind of categories of applicability related to the independency required to perform an activity or to achieve and compliance objective that a reference standard defines (e.g., the level of independence of the person performing a verification activity *mandated* in DO-178C).

2.5.4.7 RefRecommendationLevel

This class corresponds to the kind of categories of applicability related to the level of recommendation of a given activity, artefact or compliance requirement that a reference standard defines (e.g., the degree of recommendation to use the methods that ISO 26262 assigns to each ASIL within a conformity requirement).

Superclass

- *RefApplicabilitylevel*

Semantics

This Reference Recommendation Level models the kind of categories of applicability related to the level of recommendation of a given activity, artefact or compliance requirement that a reference standard defines (e.g., the degree of recommendation to use the methods that ISO 26262 assigns to each ASIL within a conformity requirement).

2.5.4.8 RefControlCategory

This class corresponds to the kind of categories of applicability related to the data control category associated to the configuration management controls placed on the data. (e.g., the CC1 and CC2 control categories defined in DO-178C).

Superclass

- *RefApplicabilitylevel*

Semantics

This Reference Control Category models the kind of categories of applicability related to the data control category associated to the configuration management controls placed on the data (e.g., the CC1 and CC2 control categories defined in DO-178C).

2.5.4.9 RefCriticalityApplicability

This class corresponds to the assignation, in a reference assurance framework, of an applicability level for a given criticality level to a reference applicability.

Attributes

- comment: *String*
  The comments that are embedded in applicability tables, which can imply constraints on the applicability specification.

Associations

- applicLevel: *RefApplicabilityLevel* [1]
  The applicability levels of the criticality applicability.
- criticLevel: *RefCriticalityLevel* [1]
  The criticality level of the criticality applicability.

Semantics

The Reference Criticality Applicability models the pair of an applicability level for a given criticality level to a RefApplicability.

2.5.4.10 RefApplicability

This class corresponds to the reference applicability tuple (composed of applicability level, criticality level and assurable element – such as a technique, a requirement or an activity) a reference requirement is composed of.

Superclass

- *NamedElement*

Attributes

- comments: *String*
  The comments that are embedded in applicability tables, which can imply constraints on the applicability specification.

Associations

- applicTarget: *ContentElement* [0..1]
  The assurable element – such as a technique, a requirement or an activity, to which a reference applicability applies to.
- applicCritic: *RefCriticalityApplicability* [0..*]
  The pair of criticality and applicability levels applied to the targeted assurable element.

Semantics
This Reference Applicability models the reference applicability tuple (composed of applicability level, criticality level and assurable element – such as a technique, a requirement or an activity) of which a reference requirement is composed.

2.5.4.11 RefApplicabilityTable

This class corresponds to the reference applicability table (composed of reference applicability tuples), which can imply constraints on the applicability specification.

Superclass
- *NamedElement*

Associations
- ownedApplicability: *RefApplicability* [0..*]
  The applicability tuples (rows) that form the table.
- tableContext: *ContentElement* [0..*]
  The elements in which the recommendation table is specified.

Semantics
This Reference Applicability Table models the reference applicability or recommendation tables from reference standards.

2.5.4.12 RefApplicabilityRel

This class corresponds to the existence of a relationship between two reference applicability specifications.

Attributes
- type: *ApplicabilityKind*
  The kind of an applicability relationship.

Associations
- source: *RefApplicability* [1]
  The reference applicability that corresponds to the source of a reference applicability relationship.
- Target: *RefApplicability* [1]
  The reference applicability that corresponds to the target of a reference applicability relationship.

Semantics
A Reference Applicability Relationship models different kinds of relationships between two reference applicability specifications. The semantics of the relationships are defined by the ApplicabilityKind enumeration.

2.5.4.13 RequirementRelKind (enumeration)

This enumeration corresponds to the possible relationships that can exist between two requirements.

Literals
- AND

Both requirements must be fulfilled.
- OR

At least one of the requirements must be fulfilled.
- XOR

Only one of the requirements can be fulfilled.
- Requires

The fulfilment of one requirement depends on the fulfilment of another requirement.
- Contributes To

Fulfilment of a requirement contributes to the fulfilment of another. This relationship also implies that the former requirements corresponds a decomposition of the latter. It is the opposite relationship to "refined to".

2.5.4.14 ApplicabilityKind (enumeration)

This enumeration corresponds to the possible relationships that can exist between two applicability specifications.

Literals

- AND
  Both applicability specifications must be fulfilled.
- OR
  At least one of the applicability specifications must be fulfilled.
- XOR
  Only one of the applicability specifications can be fulfilled.

2.5.5 Conceptual Vocabulary Metamodel

The Vocabulary Metamodel serves two principal purposes:

- It provides for the expression of vocabulary to be used in modelling the argument claims and requirements in assurance assets defined in the models of assurance frameworks to be produced at Level 1 and 1b of the AMASS framework and in the project assets to be defined at Level 2. This is achieved simply in the Vocabulary Metamodel presented here by the definition of a 'Term' concept, which captures the syntactic structure of terms and expressions used.
- It is used to provide the structure for the CACM Thesaurus, in which core terms are defined and then mapped within and across standard- and project-specific models.

The metamodel is described in Figure 20 and the subsequent subsections.

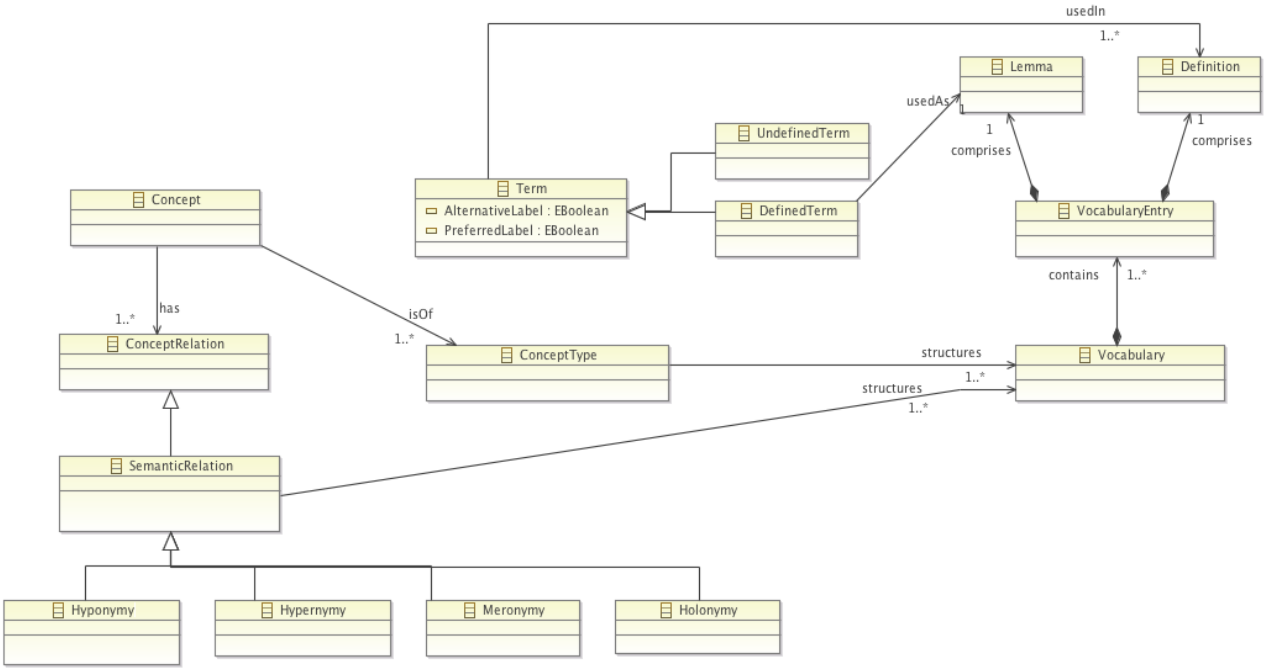


Figure 20. Conceptual Vocabulary Metamodel

2.5.5.1 Concept

Any unit of meaning which can be described or defined by a unique combination of characteristics [21]. A Concept is represented/reified by the Term class and defined by the Definition class.

2.5.5.2 ConceptType

A container class which classifies things on the basis of their similarities. Each Concept belongs to one or more ConceptType. This is akin to the "ConceptType" relation in SBVR [21]. The ConceptTypes provide the basic structure of the Vocabulary in that Terms are arranged according to the ConceptTypes of the Concepts they represent.

2.5.5.3 ConceptRelation

A container for the types of relation between Concepts, which are used to model the comparison between two Concepts in two different Vocabularies or to structure a Vocabulary in terms of the relationships between Concepts grouped within a ConceptType.

2.5.5.4 SemanticRelation

A container for the finer relationships between the meanings of Concepts within a ConceptType, used to structure Terms of the same ConceptType in a Vocabulary. Four types of SemanticRelation are defined, as follows.

2.5.5.5 Hyponymy

A SemanticRelation by which the meaning of one Concept is more specific than that of another Concept. In other words, a hyponym is used to designate a member of a general class. For example, 'systematic fault' and 'intermittent fault' are both hyponyms of 'fault'. This might be referred to as a "type-of" relationship.

2.5.5.6 Hypernymy

A SemanticRelation by which the meaning of one Concept is more general than another Concept. In other words, a hypernym designates the general category, of which its hyponyms are members or subdivisions. For example, 'fault' is a hypernym of 'systematic fault' and 'intermittent fault'. This might be referred to as a "supertype-of" relationship.

2.5.5.7 Meronymy

A SemanticRelation by which one Concept is a constituent part of a general whole captured in another Concept. For example, "wheel" is a meronym of "automobile". This might be referred to as a "part-of" relationship.

2.5.5.8 Holonymy

A SemanticRelation by which one Concept is an aggregation of other Concepts. For example, "automobile" is a holonym of "wheel", "chassis" etc. This might be referred to as a "contains" relationship.

2.5.5.9 Term

The word or phrase which represents (or reifies) a Concept, typically a noun or noun-phrase. Terms are thus the basic domain vocabulary and are stored in the Vocabulary. A Term may be thought of as providing a label for a Concept, an unambiguous means by which the Concept can be referenced. Each Term has two Boolean attributes, PreferredLabel and AlternativeLabel. If the PreferredLabel attribute is set true, then the Term serves as the primary means by which a Concept is referred to and the principal key to represent that Concept in the Vocabulary. If the AlternativeLabel is set true, the term serves as an alternate means to

reference the Concept, there being a PreferredLabel elsewhere in the Vocabulary. This mechanism allows for the unambiguous recording of exact synonym relationships between terms.

2.5.5.10 DefinedTerm and UndefinedTerm

We identify two subtypes of Term. DefinedTerms are those which are included in the Vocabulary and which should be used with a single “reserved” meaning in project artefacts. Each DefinedTerm is represented in the Vocabulary by a VocabularyEntry. UndefinedTerms are those which have no “reserved” meaning in the project, and which are not therefore defined in the Vocabulary. Terms of all kinds can be used in Definitions.

2.5.5.11 Vocabulary

An aggregation of the VocabularyEntries which identify and define DefinedTerms for a particular domain. The broad structure of the Vocabulary is provided by the ConceptTypes, and relationships between the Concepts represented by DefinedTerms are captured by SemanticRelations.

2.5.5.12 VocabularyEntry

Each DefinedTerm has a VocabularyEntry which encapsulates the information stored concerning the DefinedTerm in the Vocabulary. Each VocabularyEntry comprises exactly one Lemma and exactly one Definition.

2.5.5.13 Lemma

A string representing the noun or noun-phrased used to designate a DefinedTerm. The Lemma serves as the key to identify the DefinedTerm in the Vocabulary.

2.5.5.14 Definition

A string which contains an unambiguous description of the Concept represented by a DefinedTerm. The nouns and noun-phrases used in Definitions may be either DefinedTerms or UndefinedTerms.

2.5.6 Conceptual Mapping Definition Metamodel

The class diagram for the Standard Definition Metamodel is shown in the figure below. In the following subsections, we define the model elements.

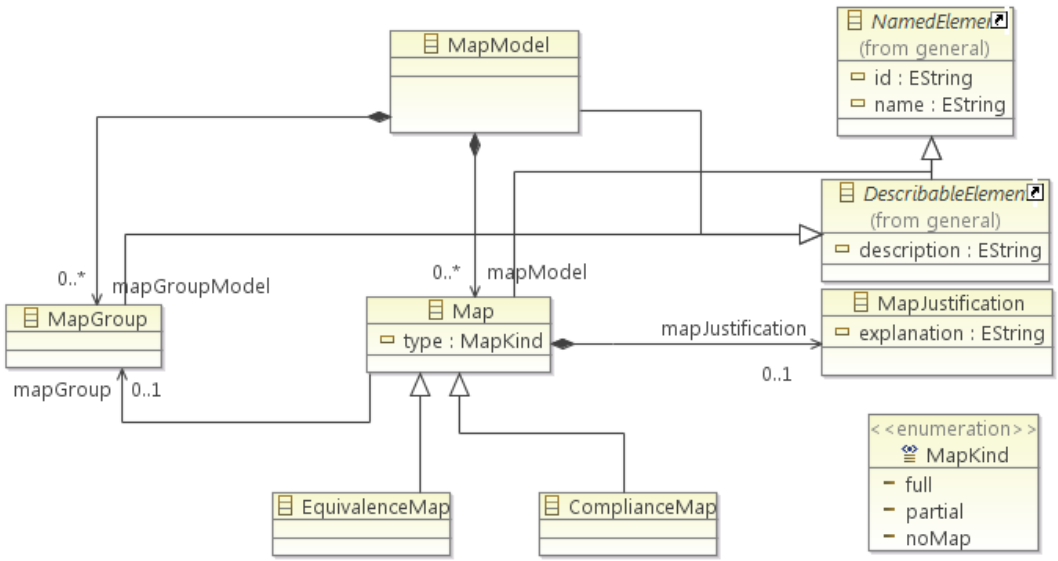


Figure 21. Mapping Definition metamodel

2.5.6.1 MapModel

This class corresponds to a model of mapping elements.

Superclass

- *DescribableElement*

Associations

- map: *Map* [0..*]
  The set of Map elements of a MapModel.
- mapGroupModel: *MapGroup* [0..*]
  The set of MapGroups that belong to the MapModel.

Semantics

A Map Model specifies the root element of a model representing a set of mapping elements.

2.5.6.2 MapGroup

This class corresponds to a group of Map elements.

Superclass

- *DescribableElement*

Semantics

A Map Group specifies a group of Map elements.

2.5.6.3 Map (Abstract)

The Map class is the container class for the types of mapping relationship. These relationships capture the similarities between elements which can be mapped to one another, and also capture the differences between them.

Superclass

- NamedElement

Attributes

- type: MapKind
  The nature of the mapping, in terms of coverage between the source and target elements.

Associations

- mapGroup: *MapGroup* [0..1]
  The MapGroup to which the Map belongs.
- mapJustification: *MapJustification* [0..1]
  A Justification text describing to which extent and under which conditions two elements map.

Semantics

The Map class is the container class for the types of mapping relationship. These relationships capture the similarities between elements which can be mapped to one another, and also capture the differences between them. A given mappable element can serve either as a source or a target in the mapping, and plural relationships are permitted by the model. It is essential for the nature of the overlaps and gaps to be detailed, in order to facilitate the expert guidance on element reuse which the AMASS Platform will seek to provide. The MapJustification attribute – which represents a note node on the mapping link – has been defined in order to provide for the recording of this detail.

2.5.6.4 EquivalenceMap

This class defines a justified (partial or total) equivalence between two DescribableElements (from UMA).

Superclass

- *Map*

Associations

- source: *DescribableElement* [0..*]
  The describable element which a map is sourcing.
- target: *DescribableElement* [0..*]
  The describable element which a map is targeting.

Semantics
An Equivalence Map models a justified equivalence between two describable elements. It is used to indicate potential (partial or full) equivalences based on the attached justification.

2.5.6.5 ComplianceMap

This class captures mapping between elements used to demonstrate compliance of a describable element (UMA) regarding reference model elements.

Superclass
- *Map*

Associations
- source: *DescribableElement* [0..*]
  The describable element which a map is sourcing.
- target: *AssuranceAsset* [0..*]
  The assurance asset which a map is targeting.

Semantics
A Compliance Map specifies the mapping between elements used to demonstrate compliance of a describable element (UMA) regarding reference model elements.

2.5.6.6 MapJustification

This class corresponds to a textual justification of the mapping type, detailing the similarities between the source and target elements and salient areas of difference. The description here should be qualitative, not quantitative.

Attributes
- explanation: String
  The placeholder for the textual justification.

Semantics
A Map Justification specifies a textual justification of the mapping type, detailing the similarities between the source and target elements and salient areas of difference. The description here should be qualitative, not quantitative.

2.5.6.7 MapKind (enumeration)

This enumeration corresponds to types of Maps that can occur in modelling mapping. It models the nature of the mapping, in terms of coverage between the source and target elements.

Literals
- full
  A full mapping represents complete coverage of the source element in the target.
- partial
  A partial mapping – which may be quantified in the implementation – indicates incomplete coverage.
- nomap
  A mapping of type 'no map' indicates that there is no similarity between the source and target elements.

2.5.6.8 DescribableElement

See definition in 2.1.2.2.

2.5.6.9 NamedElemen

See definition in 2.1.2.1.

3. Implementation CACM

3.1 General Metamodel

3.1.1 Scope and Purpose

See 2.1.1.

3.1.2 Implementation General Metamodel

The General conceptual and implementation metamodels are the same.

3.1.3 Implementation Property Metamodel

The General and Property conceptual and implementation metamodels are the same.

3.2 System Component Metamodel

3.2.1 Scope and Purpose

See 2.2.1.

3.2.2 Implementation Model Definition

The implementation of the System Component Metamodel used in AMASS has been obtained by using SysML entities and the SysML Contract profile offered by CHESS. The following sub chapters provide details about the used entities.

3.2.2.1 Modelling out of context

This part of this conceptual metamodel has been covered by the capabilities of the SysML Block and related entities to model the specific kind of system components, with owned properties and ports, and to model the system as a hierarchy of such components, with connections between them.

3.2.2.2 Contracts

This part of the conceptual metamodel has been realized through the CHESS SysML profile for contracts based design.

The entities of the CHESS Contract profile concerning modelling of contracts are represented in the following UML profile diagram.

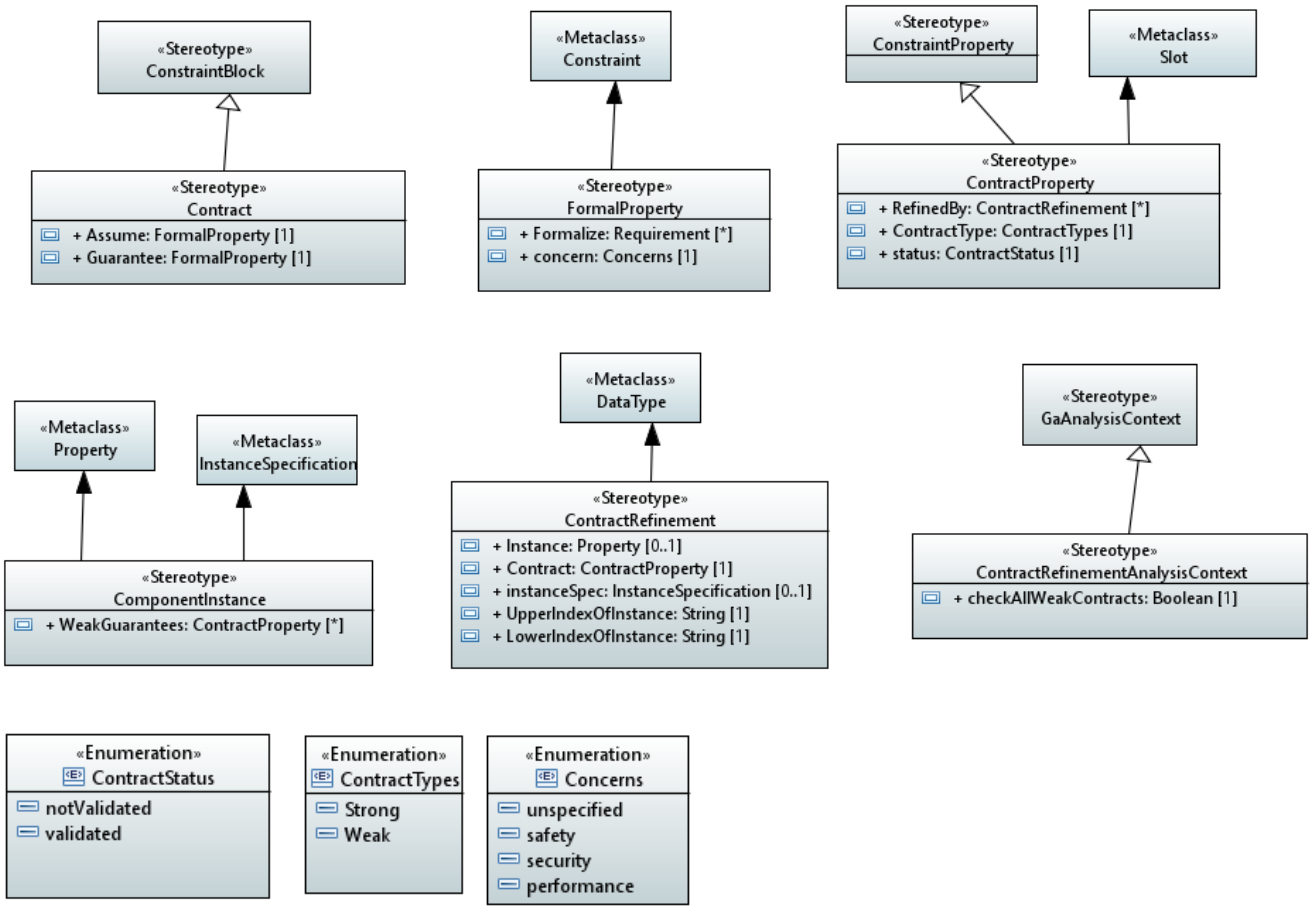


Figure 22. CHESS Contract Profile

The definition and semantic of the aforementioned entities are elaborated in the next sub-sections.

3.2.2.2.1 Contract

Contract is a stereotype which extends the SysML ConstraintBlock entity. Contract allows to aggregate two special kind of UML Constraint, in particular it allows the modelling of the assumption and guarantee contract's properties as Constraint owned by the ConstraintBlock itself.

It is worth noting that the profile does not impose any particular language to be used for the specification of the assume and guarantee contract's properties.

It is often the case that a contract is specified to put some restriction upon the attributes (i.e. their values) owned by a given component (e.g. component input and output ports): according to the SysML semantics defined for the ConstraintBlock, the attributes which are eventually subject of the assume and guarantee constraints specification (i.e. subject of the contract) can also appear as parameters (i.e. attributes) of the Contract. This allows to model contracts in isolation, so enabling their reuse, while giving the possibility to bind the contract and its constrained attributes to a given component and attributes at a later stage in the process. This is analogous to how ConstraintBlocks works in SysML (in particular by using Parametric diagram).

The association between a Contract and a component is obtained by instantiating a ContractProperty (see 3.2.2.2.3) into the component as required by SysML for the ConstraintBlock and the ConstraintProperty entities.

Extends Metaclass

None

Superclass

- SysML ConstraintBlock

Attributes

None

Relationships

- Assume : *FormalProperty* [1]
  The contract assumption
- Guarantee: *FormalProperty* [1]
  The contract guarantee

Semantics

A Contract models a contract as a pair of assumption and guarantee properties.

3.2.2.2.2 FormalProperty

FormalProperty extends the Constraint element of UML and is used to express restrictions about the possible behaviour of a component or to express the environment behaviour expected by a component; this information is typically derived starting from the available system or component requirements. The specification of a FormalProperty can be provided by using formal languages and by providing expressions upon the available component properties, e.g. its input and output ports. A FormalProperty can then appear as assumption or guarantee property of a Contract.

Extends Metaclass

None

Superclass

UML Constraint

Attributes

None

Relationships

- formalize: SysML *Requirements* [*]
  The requirements formalized by the formal property.
- concern: *Concern* [1]
  The concern addressed by the formal property.

Semantics

A FormalProperty allows to represent the concept of assumption or guarantee property.

3.2.2.2.3 ContractProperty

The ContractProperty stereotype derives from the SysML ConstraintProperty. A ContractProperty allows to bind a Contract to a component, as weak or strong contracts.

The ContractProperty stereotype has a RefinedBy attribute that refers the set of ContractProperties that decompose it. The usage of the ContractRefinement data type allows the modelling of contracts decomposition at the level of the contracts' instantiations, i.e. at the level of ContractProperties defined for the parent and child components (the latter modelled as parts, i.e. Property, in a block definition diagram).

Extends Metaclass

None

Superclass

- SysML *ConstraintProperty*

Attributes

None

Relationships

- RefinedBy: *ContractRefinement* [*]
  The properties through which the contract refinement is modelled.
- ContractType: *ContractTypes* [1]
  The kind to be considered for the contract, i.e. weak or strong.

Semantics

ContractProperty allows to bind a Contract to a given component; in particular, a component that has a property stereotyped as ContractProperty and typed with a Contract has that Contract associated. It also allows to specify if the given Contract is bound as weak or strong contract.

3.2.2.2.4 ContractRefinement

ContractRefinement is a data type used to aggregate a Property typed with a component and a ContractProperty

Extends Metaclass

UML DataType

Superclass

None

Attributes

- UpperIndexOfInstance: *String* [1]
- If the owner/s of the refining contract (that is specified in the 'Instance' relationship) is/are part of an array of components, this attribute is the upper index of the range that defines the selected owner/s in the array. LowerIndexOfInstance: String [1]
  If the owner/s of the refining contract (that is specified in the 'Instance' relationship) is/are part of an array of components, this attribute is the lower index of the range that defines the selected owner/s in the array.

Relationships

- Contract: *ContractProperty*
  The refining Contract
- Instance: UML *Property* [0..1]
  The instance (represented in the model as UML Property) owning the refining contract
- instanceSpec: UML *InstanceSpecification* [0..1]
  The instance (represented in the model as UML InstanceSpecification) owning the refining contract

Semantics

ContractRefinement represent a contract associated to a given component instance; this information is then used to model the contract refinement (see ContractProperty).

3.2.2.2.5 ComponentInstance

The ComponentInstance stereotype allows to provide the list of weak contracts which actually hold for a given component instance (see also section 3.2.2.3 about odelling of component instances).

Extends Metaclass

UML Property, UML InstanceSpecification

Superclass

None

Attributes

None

Relationships

- weakGuarantee: *ContractProperty* [*]

The weak contracts which hold for the current component instance.

Semantics

ComponentInstance allows to specify the set of weak contracts that are valid for the given component instance.

3.2.2.2.6 ContractRefinementAnalysisContext

ContractRefinementAnalysisContext allows to collect the proper information to enable contract refinement analysis.

Extends Metaclass

- MARTE GaAnalysisContext

Superclass

None

Attributes

- CheckAllWeakContract: *Boolean* [1]

  If True allows to consider weak contracts validation

Relationships

None

Semantics

Aggregates information available to enable contract refinement analysis.

3.2.2.2.7 ContractType

ContractTypes is an enumeration type that defines literals used for specifying if a given contract is a weak or strong one.

Literal values are:

- Strong: indicates a strong contract
- Weak: indicates a weak contract

3.2.2.2.8 ContractStatus

ContractStatus is an enumeration type that defines literals used for specifying the validation status of a given contract.

Literal values are:

- notValidated: indicates a contract that has not been validated
- validated: indicates a contract that has been validated

3.2.2.2.9 Concern

Concern is an enumeration type that defines literals used for specifying the concern of a given formal properties.

Literal values are:

- Unspecified
- Safety
- Security
- Performance

3.2.2.3 Modelling in a given context

Modelling in a given context addresses the possibility to represent a particular instance of a given system. An instance of a system is composed by a set of instances of components properly assembled through the

available ports. This part of the conceptual metamodel is covered by the UML InstanceSpecification construct. Typically, the system is first represented as a composite component; then a particular instance of the system is modelled by creating an InstanceSpecifications for the system component and by creating a set of owned InstanceSpecification representing the instances of the owned components (this step has to be recursively applied given that a given component can also be composite[1]). The instance level representation of the components' properties can also be created, in order to provide actual values for them.

Instances are particular useful when considering the analysis support; indeed, the analysis is performed on a specific system instance. The concept of analysis context introduced in the logical metamodel is covered by the AnalysisContext construct defined by MARTE. MARTE AnalysisContext allows to collect the information needed to run a given analysis. In particular, through the AnalysisContext it is possible to select the model entities to be analysed (e.g. a given system instance or a subset of constituent component instances) and provide additional information required by the specific analysis (like the failure condition to be analysed in case of safety analysis).

3.2.2.4 Failure Behaviour

The support for failure behaviour specification is implemented by the dependability profile defined in the CHESS modelling language. Following subsections report an excerpt of the aforementioned profile which implements the basic concepts highlighted in section 2.2.2.4. The complete description of the CHESS dependability profile is available at [20].

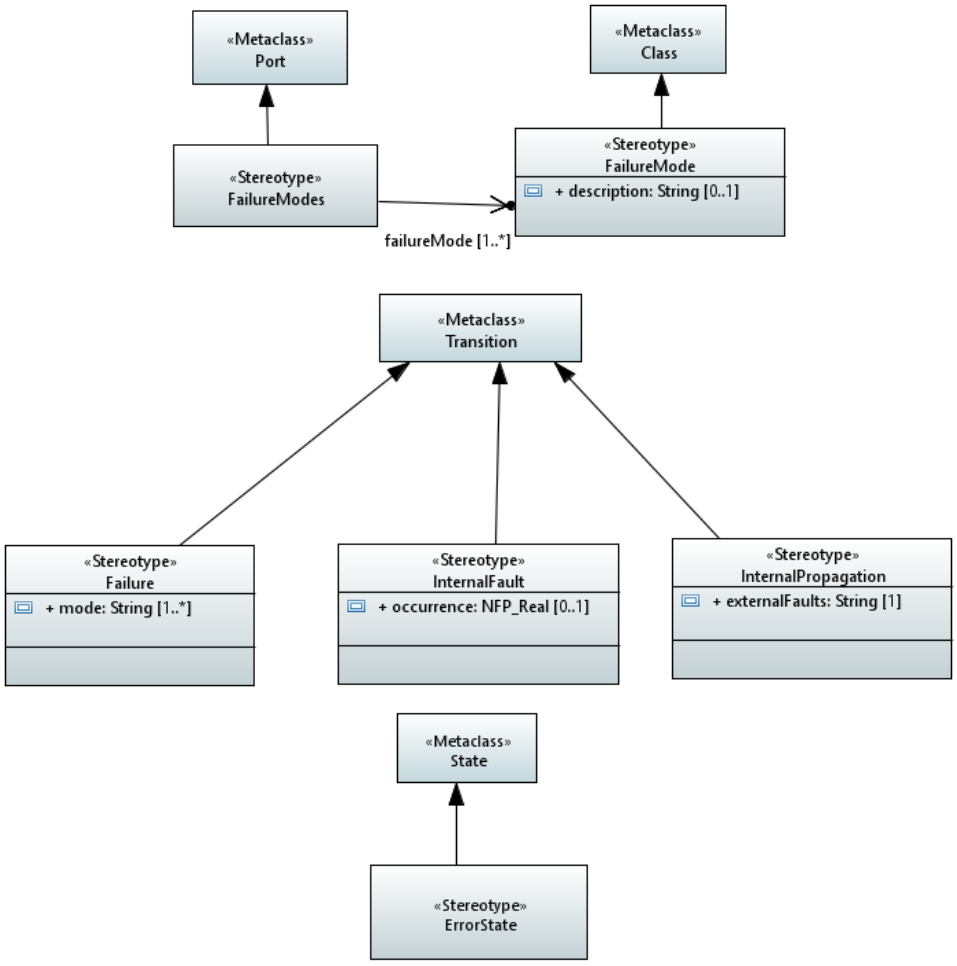


Figure 23. CHESS dependability profile excerpt

[1] The CHESS tool comes with a feature which allows to automatically derive the instance level representation of a given system

3.2.2.4.1 FailureMode

Extends Metaclass

UML Class

Superclass

None

Attributes

- Description: *String* [1]

  The description of the represented failure mode.

Relationships

None

Semantics

FailureMode represents a possible failure mode of a component.

3.2.2.4.2 FailureModes

Extends Metaclass

- UML Port

Superclass

None

Attributes

None

Relationships

- failureModes : *FailureMode* [1..*]

  The list of failure modes that can affect the given port.

Semantics

FailureModes allows to attach failure modes to component ports.

3.2.2.4.3 InternalFault

Extends Metaclass

- UML Transition

Superclass

None

Attributes

- occurrence: MARTE *NFP_Real* [0..1]

  The quantification about the possible occurrence of the internal fault.

Relationships

None

Semantics

InternalFault represents a state transition of a component caused by an internal fault.
An internal fault can bring the component to an error state.

3.2.2.4.4 InternalPropagation

Extends Metaclass

- UML Transition

Superclass

None

Attributes

- externalFault : *String* [1]
  A boolean condition about the occurrence of external failures (i.e. failure coming from the environment in input to the component); the InternalPropagation transition fires in case the externalFault condition is evaluated to true.

Relationships

None

Semantics

InternalFault represents an error propagation occurring within the component. The propagation occurs due the occurrence of failures coming from the environment (e.g. connected failed components).

3.2.2.4.5 Failure

Extends Metaclass

- UML Transition

Superclass

None

Attributes

- modes : *String* [1]
  The specification of how the component fails, i.e. which failure modes become visible on the component external ports. The specification must be provided according to the grammar described in [20].

Relationships

None

Semantics

Failure represents the occurrence of one or more failures of the component (i.e., an erroneous component state reaches the service interface, so the component ports).

3.2.2.4.6 ErrorState

Extends Metaclass

- UML State

Superclass

None

Attributes

None

Relationships

None

Semantics

ErrorState represents a state of the component that is considered erroneous (i.e., not complying with the specifications). Can have InternalFault, InternalPropagation and/or Failure has incoming/outgoing transitions.

3.2.2.5 Security Concerns

The CHESS dependability profile has been extended in AMASS to cover security aspects.

The stereotypes related to security concerns are showed in Figure 24 together with the extended base stereotypes introduced in section 3.2.2.4.

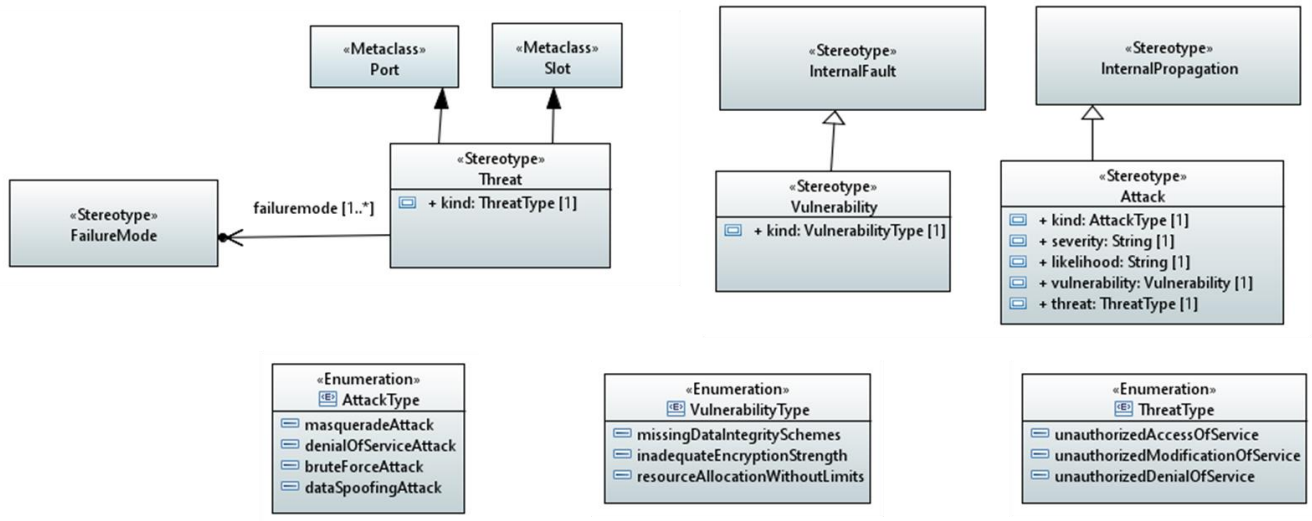


Figure 24. Security Profile

3.2.2.5.1 Attack

Extends Metaclass

None

Superclass

InternalPropagation

Attributes

- Kind: *AttackType* [1]
  The type of the attack
- Severity: *String* [1]
  The severity of the attack
- Likelihood: *String* [1]
  The likelihood of the attack
- Threat: *Threat* [1]
  The security breach caused by the attack

Relationships

- Vulnerability: Vulnerability [1]
  The vulnerability exploited by the attack.

Semantics

Attack represents an attempt to expose, alter, disable, destroy, steal or gain unauthorized access to or make unauthorized use of an asset. An attack rises due to the associated threat and cause the actual breach, if able to exploit a Vulnerability. The attack is a specialized InternalPropagation referring to the erroneous transition due to an external fault.

3.2.2.5.2 Vulnerability

Extends Metaclass

None

Superclass

InternalFault

Attributes

- Kind: *VulnerabilityType* [1]
  The type of the vulnerability

Relationships
None

Semantics
Vulnerability refers to an internal weakness of a system.

3.2.2.5.3 Threat

Extends Metaclass
UML Port, UML Slot

Superclass
None

Attributes
- Kind: *ThreatType* [1]
  The type of the threat

Relationships
None

Semantics
Threat is an event or situation that can potentially breach the security and cause harm to an asset; it affects a component port (or port instance, represented by the Slot UML metaclass).

3.2.2.5.4 AttackType

AttackType is an enumeration type that defines literals used for specifying the kind of security attacks.
Literal values are:
- masqueradeAttack
- denialOfServiceAttack
- bruteForceAttack
- dataSpoofingAttack

3.2.2.5.5 VulnerabilityType

VulnerabilityType is an enumeration type that defines literals used for specifying the kind of vulnerabilities.
Literal values are:
- missingDataIntegritySchemes
- inadequateEncryptionStrenght
- resourceAllocationWithoutLimits

3.2.2.5.6 ThreatType

ThreatType is an enumeration type that defines literals used for specifying the kind of security threats.
Literal values are:
- unauthorizedAccessOfService
- unauthorizedModificationOfService
- unauthorizedDenialOfService

3.2.2.6 Instantiate the parameterized architecture

The CHESS modelling language has been extended in AMASS to support the modelling of parameterized architectures and their instantiation. This section describes this extension.

3.2.2.6.1 InstantiatedArchitectureConfiguration

The InstantiatedArchitectureConfiguration is a stereotype which extends the Property element of UML.

Extends Metaclass

None

Superclass

- UML Property

Attributes

- ParameterList: *String* [*]
  The list of pair <ParameterName, ParameterValue>

Relationships

- InstantiatedRootComponent: *Class* [0..1]
  The optional reference to the System Block of the instantiated architecture.

3.2.2.7 Architectural Patterns

An UML profile for patterns specification and instantiation in a given system model has been defined in AMASS. This section provides the description of the stereotypes that have been defined.

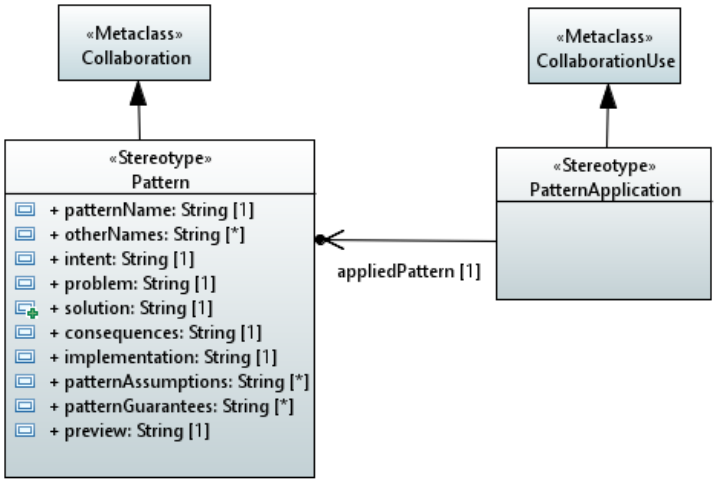


Figure 25. Pattern profile

3.2.2.7.1 Pattern

Extends Metaclass

UML Collaboration

Superclass

None

Attributes

- patternName: *String* [1]
  The name of the pattern.
- otherNames: *String* [*]
  Other names with which the design pattern can be known in different domains of application or standards.
- Intent: *String* [1]
  The context where the pattern is used. For example, define if the pattern is recommended for a specific safety-critical domain.
- Problem: *String* [1]
  The description of the problem addressed by the design pattern.
- Solution: *String* [1]

The solution to the problem under consideration. Main elements of the patterns are described.

- Consequences: *String* [1]
  The implication or consequences of using this pattern. It explains both the positive and negative implications of using the design pattern.
- Implementation: *String* [1]
  The set of points that should be taken under consideration when implementing the pattern. Language dependent.
- PatternAssumption: *String* [*]
  The contract assumptions related to the design pattern.
- PatternGuarantee: *String* [*]
  The contract guarantee related to the design pattern.
- Preview: *String* [1]
  The reference to an image showing the pattern with the collaborating entities.

Relationships

None

Semantics

Pattern represents an architectural design pattern to help designer and system architect when choosing suitable solutions for commonly recurring design problem related to functional and dependability (performance, safety, security) concerns. It extends the UML Collaboration entity.

3.2.2.7.2 PatternApplication

Extends Metaclass

UML CollaborationUse

Superclass

None

Attributes

None

Relationships

- appliedPattern: *Pattern* [1]

The pattern that has been applied.

Semantics

PatternApplication represent an instantiation of a given pattern in the context of an actual system.

3.2.2.8 Link to evidence, assurance cases and executed process

The generic approach provided by Capra [24] has been adopted to allow creating links between architectural entity to assurance case related ones, as required by the conceptual metamodel (see 2.2.2.5). The basic support for traceability provided by Capra has been extended by adding specific types of link.

The types of link used with Capra are described in the following sub-sections.

3.2.2.8.1 ContractClaimLink

Superclass

None

Attributes

None

Relationships

- sources: *Contract* [1]
- targets: *Claim* [1..*]

Semantics
A ContractClaimLink represents a specific trace connecting a component contract to claims.

3.2.2.8.2 FormalPropertyClaimLink

Superclass
None
Attributes
None
Relationships
- sources: *FormalProperty* [1]
- targets: *Claim* [1..*]

Semantics
A FormalPropertyClaimLink represents a specific trace connecting a contract assumption or guarantee to claims.

3.2.2.8.3 ContractArtefactLink

Superclass
None
Attributes
None
Relationships
- sources: *Contract* [1]
- targets: *Artefact* [1..*]

Semantics
A ContractArtefactLink represents a specific trace connecting a component contract to artefacts.

3.2.2.8.4 ContractAgreementLink

Superclass
None
Attributes
None
Relationships
- sources: *Contract* [1]
- targets: *Agreement* [1..*]

Semantics
A ContractAgreementLink represents a specific trace connecting a component contract to assurance case agreements.

3.2.2.8.5 AnalysisContextArtefactLink

Superclass
None
Attributes
None
Relationships
- sources: *AnalysisContext* [1]
- targets: *Artefact* [1..*]

Semantics
An AnalysisContextArtefactLink represents a specific trace connecting an AnalysisContext to the artefacts produced by the analysis which has been executed considering the information provided through the AnalysisContext itself.

3.2.2.8.6 ComponentArgumentationElementLink

Superclass
None

Attributes
None

Relationships
- sources: *Class* [1]
- targets: *Argumentation element* [1..*]

Semantics
A ComponentArgumentationElementLink represents a specific trace connecting a system component (represented in *sources* as UML Class, i.e. the base class for the SysML Block and UML Component constructs) to assurance case argumentation.

3.2.2.8.7 RelatedTo

Superclass
None

Attributes
None

Relationships
- sources: *EObject* [*]
- targets: EObject [*]

Semantics
It represents a generic trace link. It is also used to realize the links between the architectural model entities and the executed process ones, as presented in section 2.2.2.6.

3.3 Assurance Case Metamodel

3.3.1 Scope and Purpose

See 2.3.1.

3.3.2 Implementation Model Definition

The Implementation Assurance Case Metamodel has been split into a series of Metamodel diagrams, which are presented below (Figure 26, Figure 27 and Figure 28).

The first aspect to indicate is the modifications made on the Conceptual Argumentation Metamodel to include concepts for the modular argumentation and for patterns. The modifications try on one hand to impact the minimum as possible on the conceptual meta-model and at the same time include the concepts for the modular GSN. Some modifications have been made also with the idea to facilitate the task of implementation of the meta-model.

The changes made in order to fulfil needs for modular argumentation and patterns are highlighted in green while, the changes made in order to make it connect with other parts of the CACM metamodels are highlighted in blue.

The metamodel presented here is an extension of the Conceptual one.

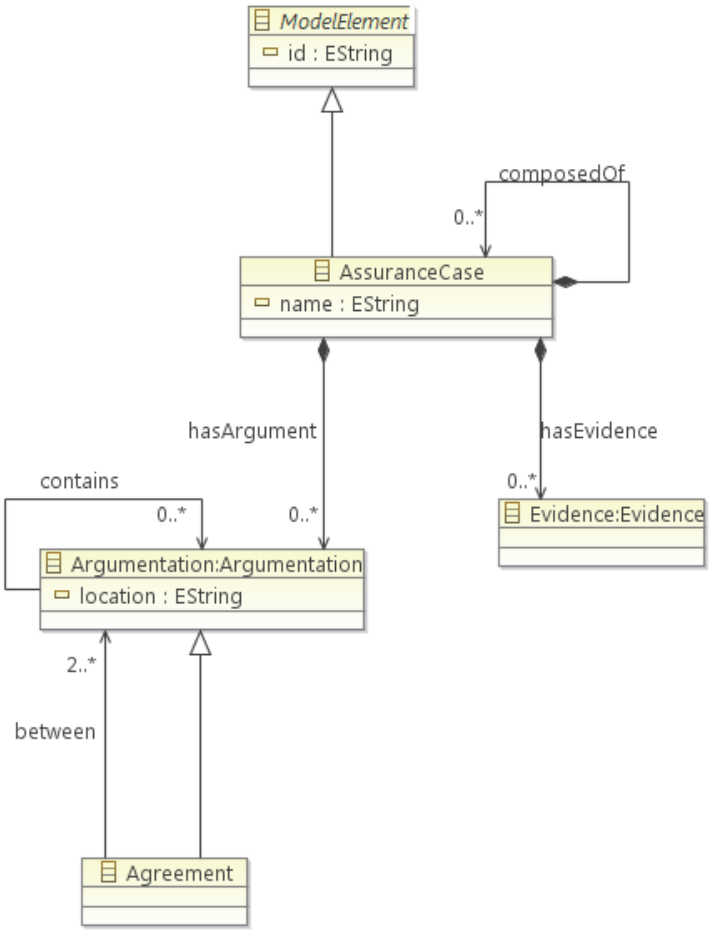


Figure 26. Assurance Case class diagram

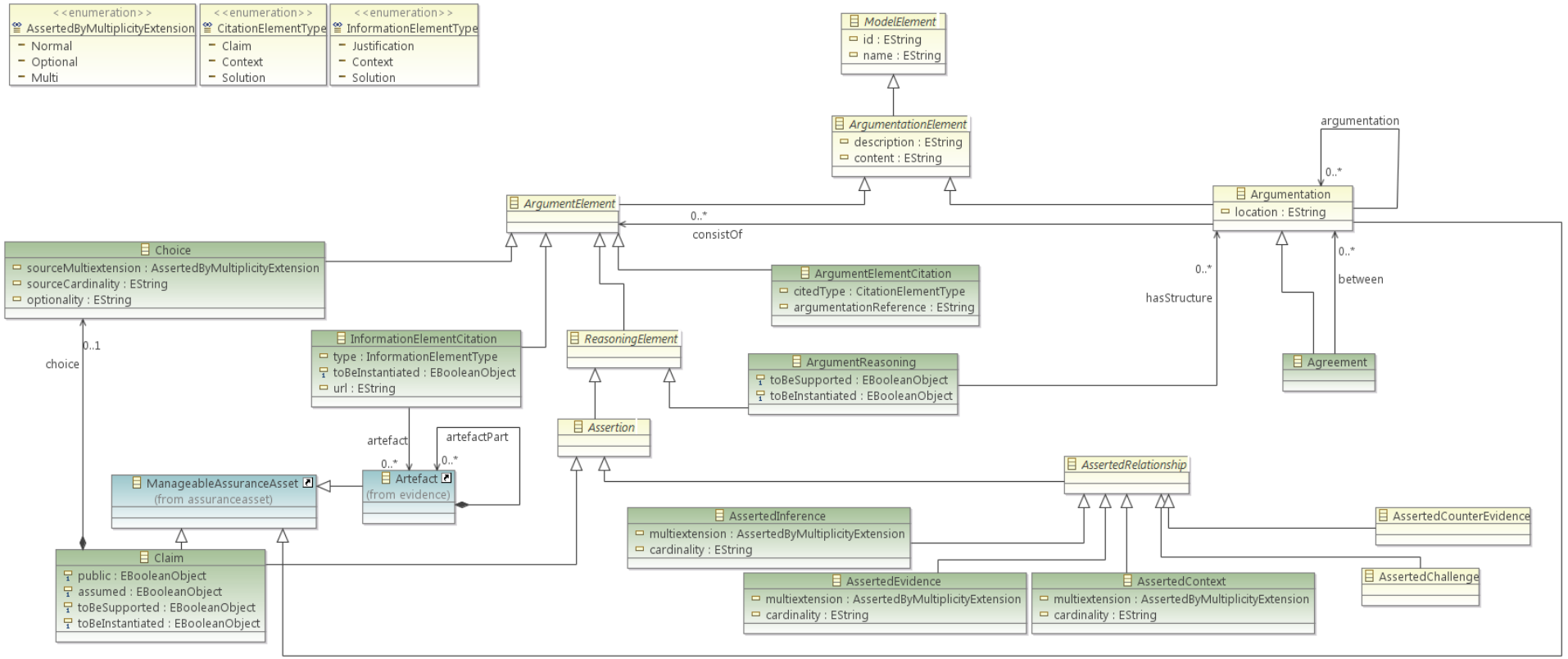


Figure 27. Argumentation Class Diagram

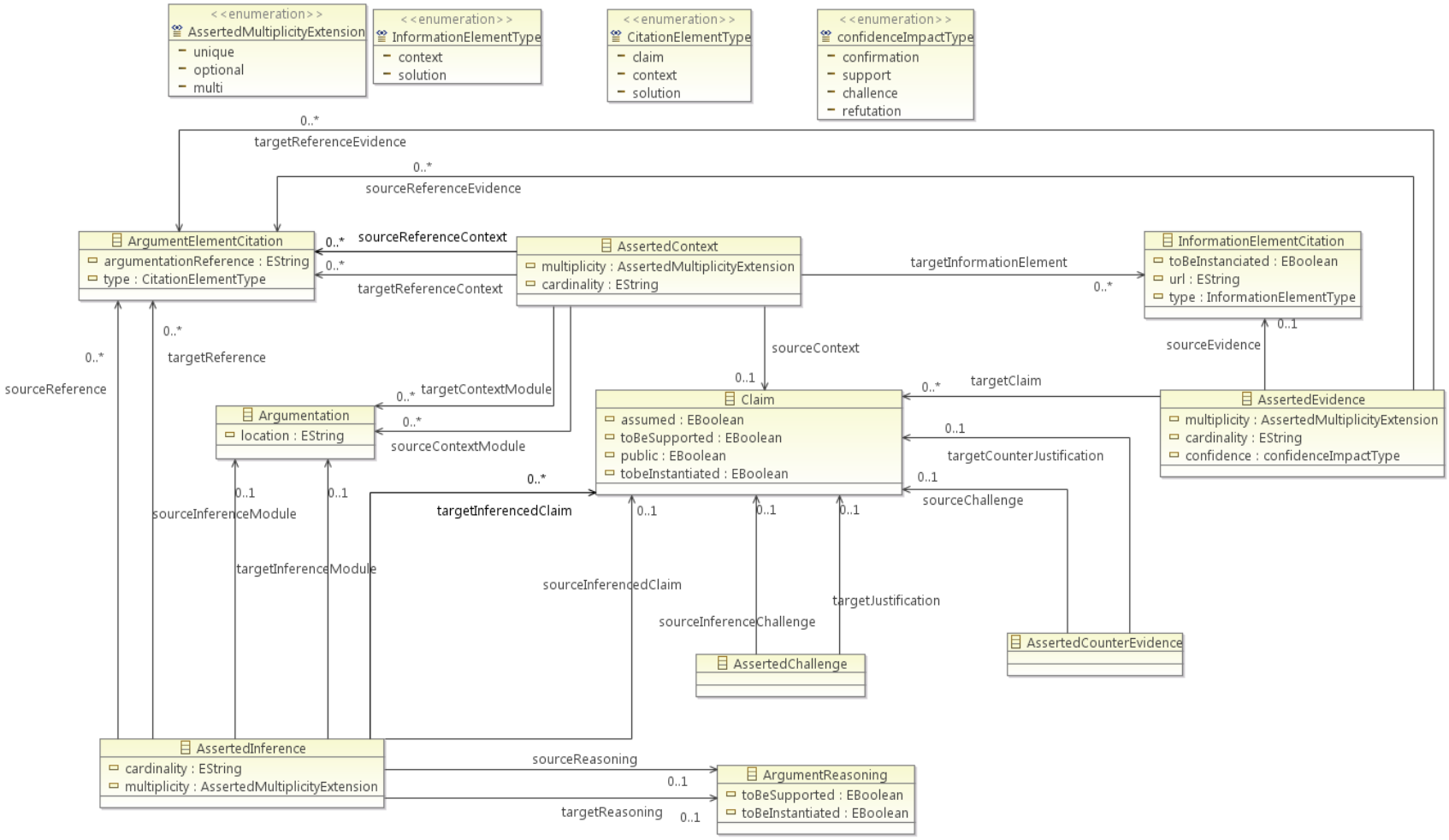


Figure 28. The Relationships view diagram

3.3.2.1 AssuranceCase

An AssuranceCase element.

Superclass
ModelElement

Attributes

- id: String
  A globally unique identified to the current assurance case
- name: String
  A comprehensive name to the current assurance case.

Associations

- hasEvidences:Evidences[0..*]
  The evidence components of the assurance case.
- hasArgument:Argumentation[0..*]
  The argument components of an assurance case.

Semantics
The AssuranceCase element represents a justified measure of confidence that a system will function as intended in its environment of use. Assurance cases can be parts of bigger assurance case. This is the case of modular approaches where an assurance case module is included on a higher-level assurance case. When a component is integrated in a system, so does its assurance case, and a component related the assurance case can be included on the system assurance case.

GraphicalNotation

None

3.3.2.2 Agreement

It is a specialisation of an AssuranceCase element.

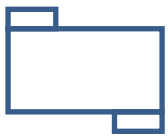
Superclass
AssuranceCase

Attributes
none
Associations

- between: Argumentation[2..*]
  The argument components, which conform the parts of the agreement.

Semantics
The Agreement element represents agreements between parts (Argumentation). Agreements are done between two or more Argumentation parts. It includes the premises and promises validated when both Argumentation are integrated.

GraphicalNotation



3.3.2.3 ArgumentationElement (abstract)

An ArgumentationElement is the top-level element of the hierarchy for argumentation elements.

Superclass
ModelElement

Attributes

- description: String

A description of the Argumentation entity.

- content: String
  Supporting content of the Argumentation entity.

Semantics

The ArgumentationElement is a common class for all elements within a structured argument.

GraphicalNotation

None

3.3.2.4 Argumentation

The Argumentation Class is the container class for a structured argument. It can be understood either as the whole argumentation of an assurance case or by an argumentation module. These modules can content another module.

Superclass

ModelElement

Attributes

- location: String
  It identifies where a module of an argumentation is stored in order to be reused.

Associations

- consistOf:ArgumentElement[0..*]
  The ArgumentElements contained in a given instance of an Argumentation.
- contains:Argumentation[0..*]
  The nested Argumentation contained in a given instance of an Argumentation.

Semantics

Structured arguments represented using the Argumentation Metamodel are composed of ArgumentElements. Argumentation elements can be nested, in this case we can talk about argumentation modules that contain elements of argumentation.

For example, arguments can be established through the composition of Claims (propositions) and the AssertedInferences between those Claims.

Another example can be seen as an argumentation module which contains a composition of Claims as before but in this case, this argumentation module is composed by a set of Claims, InformationElementCitations and/or ArgumentElementCitation.

Graphical Notation

3.3.2.5 ArgumentElement (Abstract)

The ArgumentElement Class is the abstract class for the elements of any structured argument represented using the Argumentation Metamodel.

Superclass

ArgumentationElement

Semantics

ArgumentElements represent the constituent building blocks of any structured Argument.
For example, ArgumentElements can represent the Claims and their structure made within a structured Argument.

GraphicalNotation

None

3.3.2.6 ReasoningElement (Abstract)

The ReasoningElement Class is the abstract class for the elements that comprise the core reasoning of any structured argument represented using the Argumentation Metamodel – Assertions and ArgumentReasoning (the description of inferential reasoning that exists between Claims).

Superclass
ArgumentElement

Semantics
The core of any argument is the reasoning that exists to connect assertions of that argument. Reasoning is captured in the SACM through the linking of fundamental claims and the description of the relationships between the claims. ReasoningElements represent these two elements.

GraphicalNotation
None

3.3.2.7 Assertion (Abstract)

Assertions are used to record the propositions of Argumentation (including both the Claims about the subject of the argument and structure of the Argumentation being asserted). Propositions can be true or false but cannot be true and false simultaneously.

Superclass
ReasoningElement

Semantics
Structured arguments are declared by stating claims, citing evidence and contextual information, and asserting how these elements relate to each other

GraphicalNotation
None

3.3.2.8 InformationElementCitation

The InformationElementCitation Class enables the citation of a source that relates to the structured argument. The citation is made by the InformationElementCitation class. The declaration of relationship is made by the AssertedRelationship class (an AssertedContext or an AssertedEvidence relationship).

Superclass
ArgumentElement

Attributes
- url: String
  An attribute recording a URL to external evidence.
- toBeInstantiated: Boolean
  It indicates whether the element needs to be instantiated specifically for the actual argumentation as it is part of a pattern or it just specifies the pattern.
- type: InformationElementType
  It indicates the typology of the information used.

Associations
- artefact:Artefact[0..*]
  The artefacts referenced by the current InformationElementCitation object. Artefact is a concept described in the Evidence model.

Semantics
It is necessary to be able to cite sources of information that ***support***, provide ***context*** for, or provide additional description for the core reasoning of the recorded argument. InformationElementCitations allow the citation of this information within the structured argument, thereby allowing the relationship between this information and the argument to also be explicitly declared.

The url attribute is to be used only when the argumentation aspects of the SACM are complied with. If compliance is claimed against both the argumentation and evidence packages, then the association to Evidence::Artefact shall be used to reference evidence by means of a URL.

Graphical Notation

type=”context” type=”solution”

3.3.2.9 ArgumentElementCitation

The ArgumentElementCitationt Class cites an Argumentation, or an ArgumentElement within another Argumentation, for use within the current Argumentation.

Superclass

ArgumentElement

Attributes

- type: CitationElementType
  It indicates the typology of the information used.

Associations

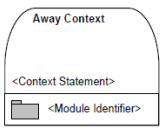
- citesElement:ArgumentElement[0..*]
  References an ArgumentElement within another Argument.

Semantics

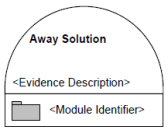
Within the actual Argumentation (package), it is sometimes useful to be able to cite elements of another Argumentation (i.e., ArgumentElements). For example, in supporting a Claim it may be useful to cite a Claim or InformationElementCitation declared within another Argumentation. It can also be useful to be able to cite entire Argumentations. For example, in supporting a Claim it may be useful to cite an existing (structured) Argumentation.

This concept is key to understand the modular argumentation. There are times when it becomes necessary to be able to make a reference from the argument of one case module to some defined context that exists within the boundary of another, or to a Claim that is supported within another argumentation structure.

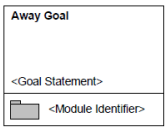
Graphical Notation



type=”context”



type=”solution”



type=”claim”

3.3.2.10 Claim

Claims, maps with 2.3.2.5, are used to record the propositions of any structured Argumentation. Propositions are instances of statements that could be true or false but cannot be true and false simultaneously.

Superclass

Assertion

ManagableAssuranceAsset (from AssuranceAsset model)

Attributes

- assumed: Boolean
  An attribute recording whether the claim being made is declared as being assumed to be true rather than being supported by further reasoning

- toBeSupported: Boolean
  An attribute recording whether further reasoning has yet to be provided to support the Claim (e.g., further evidence to be cited).
- public: Boolean
  An attribute recording whether the preposition described in the claim is publicly visible to other arguments and this way is able to be references in other structures of argumentation.
- toBeInstantiated: Boolean
  An attribute recording whether the claim needs to be instantiated for the actual argumentation or is just the specification of a pattern

Associations

- choice:Choice[0..1]
  References a ChoiceElement. A claim can be decomposed in a choice of options

Semantics

The core of any argument is a series of claims (premises) that are asserted to provide sufficient reasoning to support a (higher-level) claim (i.e., a conclusion).

A Claim that is intentionally declared without any supporting evidence or argumentation can be declared as being assumed to be true. It is an assumption. However, it should be noted that a Claim that is not 'assumed' (i.e., assumed = false) is not being declared as false.

A Claim that is intentionally declared as requiring further evidence or argumentation can be denoted by setting toBeSupported to be true.

Claims are related with InformationElementCitation through the AssertedEvidence relationship. Claims are also related to another claim in a decomposition structure. The AssertedInference relationship is also used to refer to such relationships.

Also, if a claim is referenced by a CitedElement, then this is done by the AssertedInference relationship. This is the case in modular argumentation when in an argumentation module a claim described in another argumentation module is cited. In this case the claim reference should have the public attribute equal true.

Invariants

Self.assumed and self.toBeSupported cannot both be true simultaneously

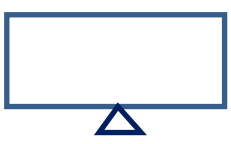
Graphical Notation

assumed=false
toBeSupported=false
toBeInstantiated=false

assumed=true
toBeSupported=false
toBeInstantiated=false

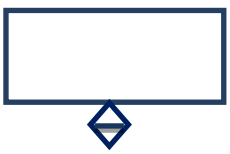
assumed=false
toBeSupported=true
toBeInstantiated=false

assumed=false
toBeSupported=false
toBeInstantiated=true



assumed=false
toBeSupported=false
toBeInstantiated=false
public=true

assumed=false
toBeSupported= true
toBeInstantiated=true



3.3.2.11 EvidenceUseAssertion

A sub-type of Claim used to record propositions (assertions) made ***regarding*** an InformationElementCitation ***being used as supporting evidence*** to the Argument. This is intended to be used as an interface element to external evidence. An evidence use assertion is a minimal assertion

(proposition) about an item of evidence, and there is no supporting argumentation being offered within the current structured argument.

Superclass
Claim

Semantics
Well-supported arguments are those where evidence can be cited that is said to support the most fundamental claims of the argument. It is good practice that these fundamental claims of the argument state clearly the property that is said to exist in, be derived from, or be exhibited by the cited evidence. Where such claims are made these are said to be basic EvidenceUseAssertions.

3.3.2.12 ArgumentReasoning

ArgumentReasoning can be used to provide additional description or explanation of the asserted inference or challenge that connects one or more Claims (premises) to another Claim (conclusion). ArgumentReasoning elements are therefore related to AssertedInferences and AssertedChallenges. It is also possible that ArgumentReasoning elements can refer to other structured Arguments as a means of documenting the detail of the argument that establishes the asserted inferences.

Superclass
ReasoningElement

Attributes
- toBeSupported: Boolean
  An attribute recording whether further reasoning has yet to be provided to support the reasoning (e.g., further evidence to be cited).
- toBeInstantiated: Boolean
  An attribute recording whether the reasoning needs to be instantiated for the actual argumentation of is just the specification of a pattern

Associations
- hasStructure:Argument[0..1]
  Optional reference to another structured Argument to provide the detailed structure of the Argument being described by the ArgumentReasoning.

Semantics
The argument step that relates one or more Claims (premises) to another Claim (conclusion) may not always be obvious.

In such cases ArgumentReasoning can be used to provide further description of the reasoning steps involved.

An ArgumentReasioning can be related with an InformationElementCitation through the AssertedContext relationship.

Graphical Notation

3.3.2.13 AssertedRelationship (Abstract)

The AssertedRelationship Class is the abstract association class that enables the ArgumentElements of any structured argument to be linked together. The linking together of ArgumentElements allows a user to declare the relationship that they assert to hold between these elements.

Superclass
Assertion

Associations

- hasSource:ArgumentationElement[0..*]
  Reference to the ArgumentationElement(s) that are the source (start-point) of the relationship.
- hasTarget:ArgumentationElement[0..*]
  Reference to the ArgumentationElement(s) that are the target (end-point) of the relationship.

Semantics

In the SACM, the structure of an argument is declared through the linking together of primitive ArgumentElements. For example, a sufficient inference can be asserted to exist between two claims ("Claim A implies Claim B") or sufficient evidence can be asserted to exist to support a claim ("Claim A is evidenced by Evidence B"). An inference asserted between two claims (A – the source – and B – the target) denotes that the truth of Claim A is said to infer the truth of Claim B.

3.3.2.14 AssertedInference

The AssertedInference, joined with AssertedRelationship maps with 2.3.2.2, association class records the inference that a user declares to exist between one or more Assertion (premises) and another Assertion (conclusion) or between Argument modules in order to define the argumentation architecture or structure. It is important to note that such a declaration is itself an assertion on behalf of the user.

Superclass

AssertedRelationship

Attributes

- multiplicity: AssertedMultiplicityExtension
  An attribute used while specifying patterns to indicate whether the inference is multiple, optional or one to one.
- cardinality: String
  An attribute used while specifying patterns to record the number of times the inference should be instantiated afterwards.

Semantics

The core structure of an argument is declared through the inferences that are asserted to exist between Assertions (e.g., Claims). For example, an AssertedInference can be said to exist between two claims ("Claim A implies Claim B"). An AssertedInference between two claims (A – the source – and B – the target) denotes that the truth of Claim A is said to infer the truth of Claim B.

An AssertedInference can relate a claim with another claim for example for decomposition needs.

An AssertedInference can relate a claim with an ArgumentElementCitation when that cited element references a Claim in another argumentation structure or module.

Invariants

context AssertedInference
inv SourceMustBeClaimOrArgumentElementCitation: self.source->forAll(s|s.oclIsTypeOf(Claim)) or t.oclIsTypeOf(InformationElementCitation))
inv TargetMustBeClaimOrAssertedRelationshipOrArgumentElementCitation: self.target -> forAll(t|t.oclIsTypeOf(Claim) or t.oclIsTypeOf(AssertedRelationship) or t.oclIsTypeOf(ArgumentElementCitation))

Graphical Notation

multiplicity=normal multiplicity=optional multiplicity=multi

3.3.2.15 Choice

This class, maps with 2.3.2.3, is a subtype of the AssertedInference Class. It is used to denote possible alternatives in satisfying an inference.

Superclass
AssertedInference

Attributes

- sourceMultiextension: AssertedMultiplicityExtension
  An attribute used while specifying patterns to indicate whether the source of the inference is multiple, optional or one to one.

Semantics
It is used to denote possible alternatives in satisfying an inference. It can represent 1-of-n and m-of-n selection, an annotation indicating the nature of the choice to be made.

Graphical Notation

3.3.2.16 AssertedEvidence

The AssertedEvidence, joined with AssertedRelationship maps with 2.3.2.1, association class records the declaration that one or more items of Evidence (cited by InformationItems). It provides information that helps establish the truth of a Claim. It is important to note that such a declaration is itself an assertion on behalf of the user. The information (cited by an InformationItem) may provide evidence for more than one Claim.

Superclass
AssertedRelationship

Attributes

- multiplicity: AssertedMultiplicityExtension
  An attribute used while specifying patterns to indicate whether the inference is multiple, optional or one to one.
- cardinality: String
  An attribute used while specifying patterns to record the number of times the inference should be instantiated afterwards.

Semantics
Where evidence (cited by InformationItems) exists that helps to establish the truth of a Claim in the argument, this relationship between the Claim and the evidence can be asserted by an AssertedEvidence association. An AssertedEvidence association between some information cited by an InformationElementCitation and a Claim (A – the source evidence cited – and B – the target claim) denotes that the evidence cited by A is said to help establish the truth of Claim B.

An AssertedEvidence can relation an InformationElementCitation with a Claim.

Invariants
context AssertedEvidence
inv SourceMustBe InformationElementCitation: self.source->
forAll(s|s.oclIsTypeOf(InformationElementCitation))
inv TargetMustBeClaimOrAssertedRelationship: self.target->forAll(t|t.oclIsTypeOf(Claim) or
t.oclIsTypeOf(AssertedRelationship))

Graphical Notation
multiplicity=normal multiplicity=optional multiplicity=multi

3.3.2.17 AssertedContext

The AssertedContext, maps with 2.3.2.4, association class declares that the information cited by an InformationElementCitation provides a context for the interpretation and definition of a Claim or ArgumentReasoning element.

Superclass
AssertedRelationship

Attributes

- multiplicity: AssertedMultiplicityExtension
  An attribute used while specifying patterns to indicate whether the context reference is multiple, optional or one to one.
- cardinality: String
  An attribute used while specifying patterns to record the number of times the context should be instantiated afterwards.

Semantics
Claim and ArgumentReasoning often need contextual information to be cited in order for the scope and definition of the reasoning to be easily interpreted. For example, a Claim can be said to be valid only in a defined context ("Claim A is asserted to be true only in a context as defined by the information cited by InformationItem B" or conversely "InformationItem B is the valid context for Claim A"). A declaration (AssertedContext) of context (InformationItem) for a ReasoningElement (A – the contextual InformationItem – and B – the ReasoningElement) denotes that A is asserted to be valid contextual information for B (i.e., A defines context where the reasoning presented by B holds true).

An AssertedContext can relation an InformationElementCitation with a ReasoningElement.

An AssertedContext can relation an InformationElementCitation with an ArgumentElementCitation when that cited element references a ReasoningElement in another argumentation structure or module.

Invariants
context AssertedContext
inv SourceMustBeInformationElementCitation: self.source->
forAll(s|s.oclIsTypeOf(InformationElementCitation))
inv TargetMustBeReasoningElementOrArgumentElementCitation: self.target ->
forAll(t|t.oclIsTypeOf(ReasoningElement) or t.oclIsTypeOf(ArgumentElementCitation))

Graphical Notation
multiplicity=normal multiplicity=optional multiplicity=multi

3.3.2.18 AssertedChallenge

The AssertedChallenge association class records the challenge (i.e., counter-argument) that a user declares to exist between one or more Claims and another Claim. It is important to note that such a declaration is itself an assertion on behalf of the user.

Superclass
AssertedRelationship

Semantics
An AssertedChallenge by Claim A (source) to Claim B (target) denotes that the truth of Claim A challenges the truth of Claim B (i.e., Claim A leads towards the conclusion that Claim B is false). This concept is used in a review process in order to indicate the weakness of an assertion associated with a claim.

Invariants
context AssertedChallenge
inv SourceMustBeClaim : self.source->forAll(s|s.oclIsTypeOf(Claim))
inv TargetMustBeClaimOrAssertedRelationship : self.target->forAll(t|t.oclIsTypeOf(Claim) or

t.oclIsTypeOf(AssertedRelationship))

Graphical Notation

3.3.2.19 AssertedCounterEvidence

AssertedCounterEvidence can be used to associate evidence (cited by InformationElements) to a Claim, where this evidence is being asserted to infer that the Claim is *false*. It is important to note that such a declaration is itself an assertion on behalf of the user.

Superclass
AssertedRelationship

Semantics
An AssertedCounterEvidence association between some evidence cited by an InformationNode and a Claim (A – the source evidence cited – and B – the target claim) denotes that the evidence cited by A is counter-evidence to the truth of Claim B (i.e., Evidence A suggests the conclusion that Claim B is false).

Invariants
context AssertedCounterEvidence
inv SourceMustBeInformationElement : self.source->forAll(s|s.oclIsTypeOf(InformationElement))
inv TargetMustBeClaimOrAssertedRelationship : self.target->forAll(t|t.oclIsTypeOf(Claim) or t.oclIsTypeOf(AssertedRelationship))

3.3.2.20 ManageableAssuranceAsset

See definition in 3.4.3.3.

3.3.2.21 Artefact

Maps with 2.3.2.6. See its definition in 3.4.3.3.

3.4 Evidence Management Metamodels

3.4.1 Scope and Purpose

See 2.4.1.

3.4.2 Implementation Traceability Metamodel (AssuranceAsset)

From the implementation point of view, the traceability of evidences is covered by the Assurance Asset Metamodel that provides classes for the recording of lifecycle events associated with these assets, and for the record of rationale and judgements concerning them.

The concept of an Assurance Asset (i.e. an assurance asset whose qualities and lifecycle can be recorded and analysed) is identified, to handle the tangible assurance assets defined in the Assurance Project Metamodel.

The traceability of the evidences is managed partially also inside this metamodel by means of the element 3.4.3.6.

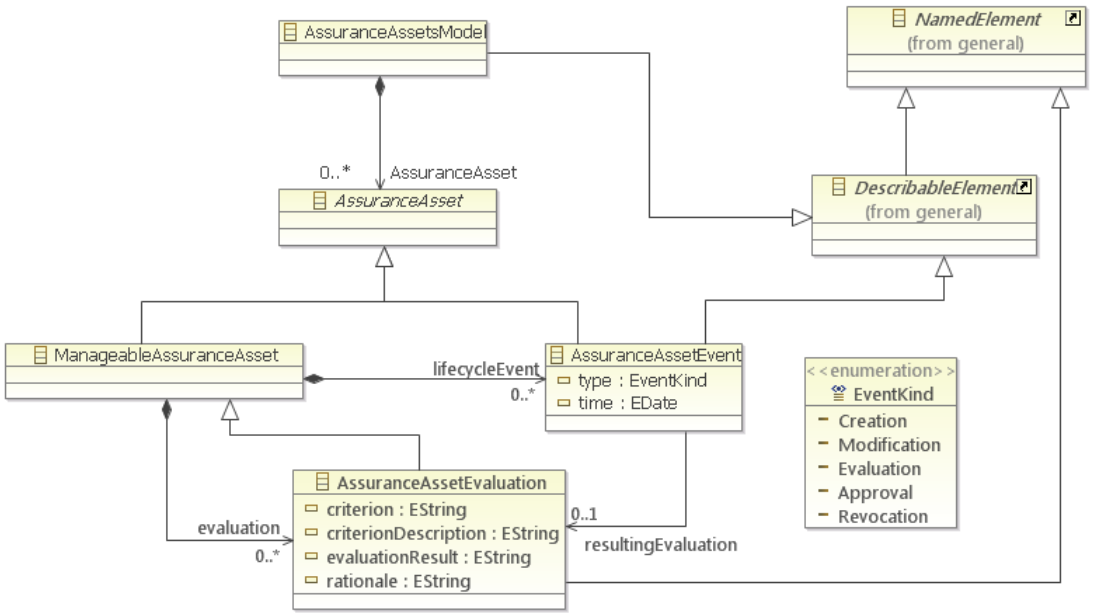


Figure 29. Assurance Asset Metamodel

3.4.2.1 AssuranceAssetsModel

This class corresponds to model of assurance assets which can be part of an assurance project.

Superclass

- *DescribableElement*

Relationships

- assuranceAsset: *AssuranceAsset* [0..*]
  The set of assurance assets that are part of the AssuranceAssetsModel

Semantics

An Assurance Assets Model represents the root model element to create assurance assets.

3.4.2.2 AssuranceAsset (abstract)

This class corresponds to an assurance asset whose qualities and lifecycle can be recorded and analysed.

Attributes

None

Relationships

None

Semantics

An Assurance Asset models any asset from assurance projects whose qualities and lifecycle can be recorded and analysed (e.g., an artefact or an activity).

3.4.2.3 ManageableAssuranceAsset

This class corresponds to assurance assets that can be evaluated and whose lifecycle might have to be recorded.

Superclass

- *AssuranceAsset*

Relationships

- evaluation: *AssuranceAssetEvaluation* [0..*]
  The assurance asset evaluations that specify the outcome of evaluating a manageable assurance asset.

- lifecycleEvent: *AssuranceAssetEvent* [0..*]
  The assurance asset events of which the lifecycle of a manageable assurance asset consists.

Semantics
A Manageable Assurance Asset models any assurance assets that can be evaluated and whose lifecycle might have to be recorded.

3.4.2.4 AssuranceAssetEvaluation

This class corresponds to the specification of the result of making some judgement regarding a manageable assurance asset.

Superclass
- *ManageableAssuranceAsset*
- *NamedElement*

Attributes
- criterion: String
  The criterion used to evaluate a manageable assurance asset.
- criterionDescription: String
  The description of the criterion used to evaluate a manageable assurance asset.
- evaluationResult: String
  The result that is specified for the evaluation of a manageable assurance asset.
- rationale: String

Semantics
An Assurance Asset Evaluation models any result of making some judgement regarding a manageable assurance asset.

3.4.2.5 AssuranceAssetEvent

This class corresponds to relevant happenings in the lifecycle of a manageable assurance asset. This serves to maintain a history log for assurance assets.

Superclass
- *AssuranceAsset*
- *DescribableElement*

Attributes
- type: *EventType*
  The type of happening of an assurance asset event.
- time: Date
  The time when an assurance asset event occurred.

Relationships
- resultingEvaluation: *AssuranceAssetEvaluation* [0..1]
  The assurance asset evaluation in which an assurance asset event results.

Semantics
An Assurance Asset Evaluation models any relevant happenings in the lifecycle of a manageable assurance asset. This serves to maintain a history log for assurance assets.

3.4.2.6 EventKind (enumeration)

This enumeration corresponds to types of events that can occur in the lifecycle of a manageable assurance asset [7, 12].

Literals
- Creation
  When a manageable assurance asset is brought into existence.
- Modification

When a change is made in some characteristic of a manageable assurance asset.

- Evaluation
  When a manageable assurance asset is evaluated.
- Approval
  When a manageable assurance asset is approved (as valid).
- Revocation
  When a manageable assurance asset is revoked.

3.4.3 Implementation Managed Artefact Metamodel

From the implementation perspective, the metamodel for Artefacts is the OPENCOSS Artefact Metamodel [13] without any of the adaptations mentioned in 2.4.3.

The class diagram for the Artefact Model is presented in the figure below.

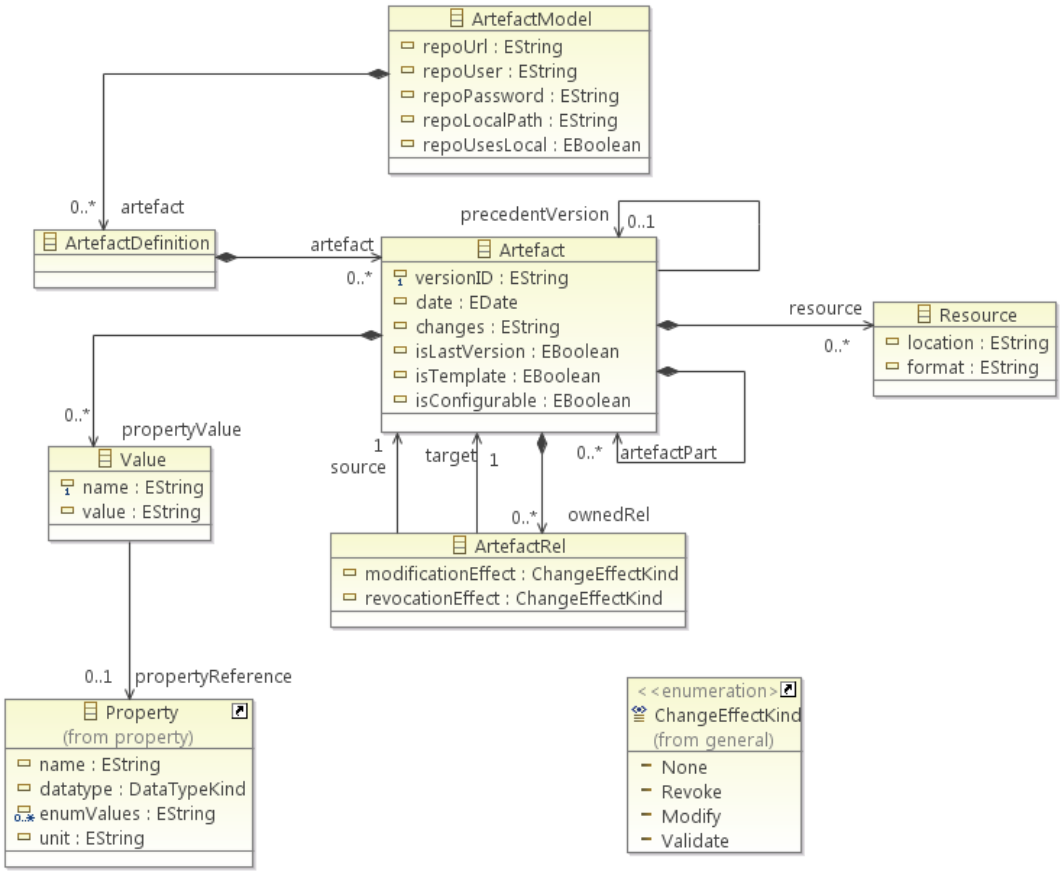


Figure 30. Artefact Metamodel (Part 1: Core Model Elements)

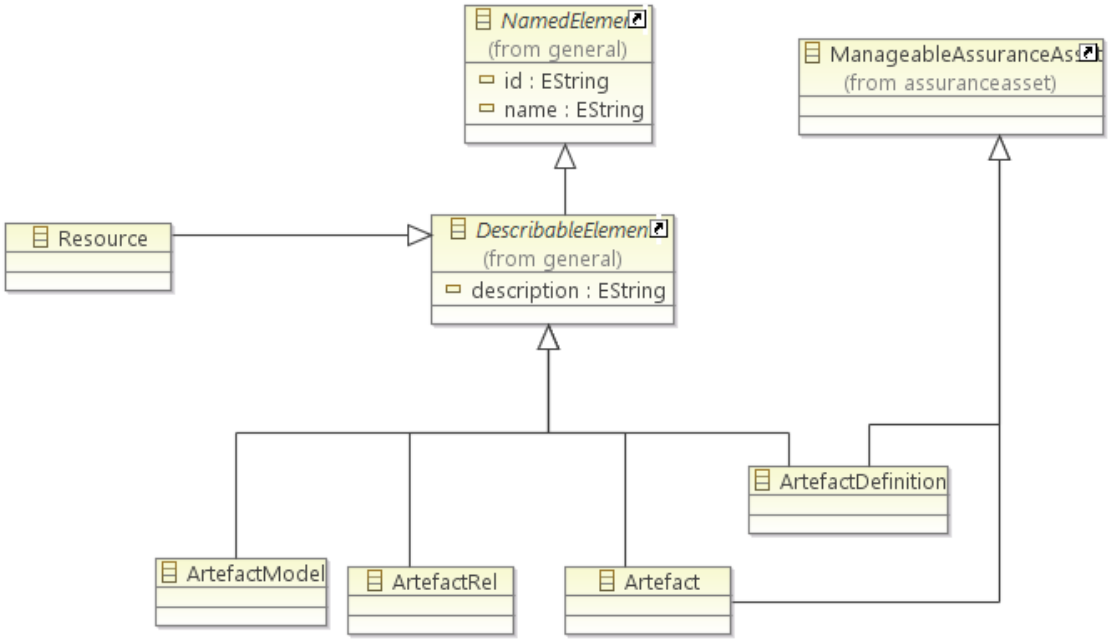


Figure 31. Artefact Metamodel (Part 2: Inheritance Relationships)

3.4.3.1 ArtefactModel

This class corresponds to a model of artefacts which are used in a given assurance project. This element maps with the ManagedArtifactModel metaclass from the conceptual view.

Superclass

- *DescribableElement*
- *ManegeableAssuranceAsset*

Attributes

- repoUrl: *String*
  The URL of a SVN repository where the Artefact resources are stored.
- repoUser: *String*
  The User name of a SVN repository where the Artefact resources are stored.
- repoPassword: *String*
  The Password of a SVN repository where the Artefact resources are stored.
- repoLocalPath: *String*
  The local path of a local repository (alternative to SVN repository) where the Artefact resources are stored.
- repoUsesLocal: *Boolean*
  A flag indicating if the Artefacts of the Artefact Model are stored in a local repository instead of a SVN repository.

Relationships

- artefact: *ArtefactDefinition* [0..*]
  The set of Artefact Definitions that belongs to the Artefact Model.

Semantics

An Artefact Model specifies the root element of a model representing a set of Artefacts. This concept embeds the access parameters to the repository of artefact resources (e.g., files). Currently, the parameters are related to two kind of artefact repositories: SVN or local hard disk repositories.

3.4.3.2 ArtefactDefinition

This class corresponds to a distinguishable abstract unit of data to manage in an assurance project that depicts the whole lifecycle resulting from the evolution, in different versions, of Artefacts.
This element maps with the ManagedArtifact from the conceptual view.

Superclass

- DescribableElement

Associations

- artefact:Artifact [0..*]
  The Artifacts of the ArtefactDefinition

Semantics

The artefacts managed in an assurance project can evolve during the project.

3.4.3.3 Artefact

This class corresponds to an artefact instance which is part of a given assurance project. An Artefact has concrete objects (called Resources), which correspond to tangible files or other information resources. This element maps with the ManagedArtifact metaclass from the conceptual view.

Superclass

- *DescribableElement*
- *ManegeableAssuranceAsset*

Attributes

- versionID: *String*
  The version number or ID of the Artefact.
- date: *Date*
  The date of creation of the current Artefact version.
- changes: *String*
  The list of changes describing any update regarding the previous Artefact version.
- isLastVersion: *Boolean*
  A flag to indicate if the Artefact version is the last one.
- isTemplate: *Boolean*
  A flag indicating if the Artefact is a Template to create the actual Artefact to be used in the assurance project.
- isConfigurable: *Boolean*
  A flag indicating if the Artefact can be configured for specific usage contexts or situations.

Relationships

- artefactPart: *Artefact* [0..*]
  The part of the Artefact which can represent document sections or any element composing the whole Artefact.
- precedentVersion: *Artefact* [0..1]
  A pointer to the precedent version of an Artefact.
- resource: *Resource* [0..*]
  The Resource elements which represente tangible objects of an artefact. For instance, the set of architectural model files of an Architecture Design document.
- propertyValue: *Value* [0..*]
  A set of attributes and their values characterising an Artefact (e.g. Confidence properties).
- ownedRel: *ArtefactRel* [0..*]
  The artefact relationships owned by an artefact.

Semantics

An Artefact specifies the instance of artefacts characterised for a version and a set of resources modelling tangible artefact resources. Artefacts are subject to traceability for change management and to

characterisation by means of property values. An Artefact can be composed of other artefacts or artefact parts.

3.4.3.4 Resource

This class corresponds to a model of a tangible object representing the artefact and maps with the Resource metaclass from the conceptual view.

Superclass

- *Describable Element*

Attributes

- location: *String*
  The path or URL string specifying the location of the resource.
- format: *String*
  The format of the resource (e.g., MS Word).

Semantics

A Resource models' tangible objects representing the Artefact, such as files or other electronic resource.

3.4.3.5 Value

This class corresponds to the value of an attribute of an artefact. This element joined the Property class maps with the ManagedArtifactProperty metaclass from the conceptual view.

Attributes

- name: *String*
  The name of the attribute value of an artefact for which a value is specified (for instance, average confidence).
- value: *String*
  The value of an attribute of an artefact (i.e., the property).

Relationships

- propertyReference: *Property [0..1]*
  The attribute of an artefact for which a value is specified. An attribute corresponds to objective, factual characteristic of an artefact [11, 10, 12].

Semantics

A Value models the value of an artefact's property. It can represent a maximum, minimum, average, etc. value. This information must be indicated in the name of the value.

3.4.3.6 ArtefactRel

This class corresponds to the existence of a relationship between two artefacts. This is the main mechanism for establishing bilateral traceability between artefacts, for example the relationship by which a test verifies a requirement and a requirement is verified by a test.

Superclass

- *Describable Element*

Attributes

- modificationEffect: *ChangeEffectKind*
  The effect that the modification of the target of an artefact relationship has on the source of the artefact relationship.
- revocationEffect: *ChangeEffect*
  The effect that the revocation of the target of an artefact relationship has on the source of the artefact relationship.

Relationships

- target: *Artefact* [1]

The artefact that correspond to the target of an artefact relationship.

- source: *Artefact* [1]
  The artefact that correspond to the source of an artefact relationship.

Semantics

An Artefact Relationship models the relationship between two artefacts. This is the main mechanism for establishing bilateral traceability between artefacts, for example the relationship by which a test verifies a requirement and a requirement is verified by a test.

3.4.3.7 Property

See definition in 2.1.3.2.

3.4.3.8 ChangeEffectKind

See definition in 2.1.2.4.

3.4.3.9 DescribableElement

See definition in 2.1.2.2.

3.4.3.10 NamedElement

See definition in 2.1.2.1.

3.4.4 Implementation Executed Process Metamodel

The implementation metamodel is basically the same as the conceptual one. The main differences are the name of some elements, the name of their relationships with other elements and the inheritance of some elements.

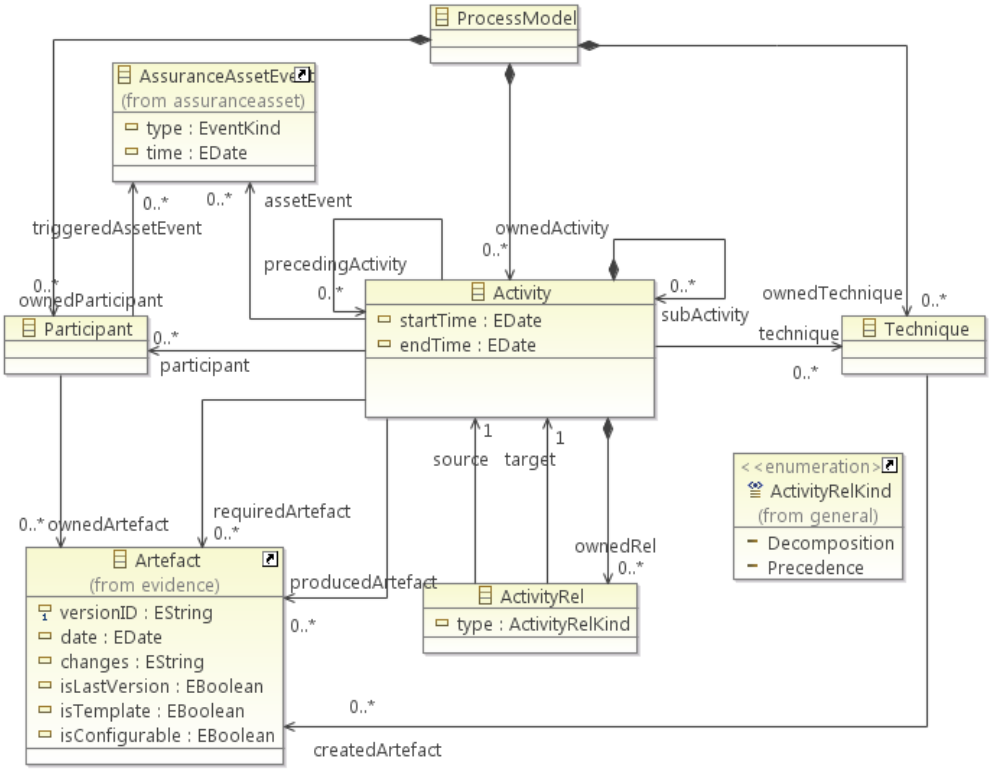


Figure 32. Process Metamodel (Part 1: Core Model Elements)

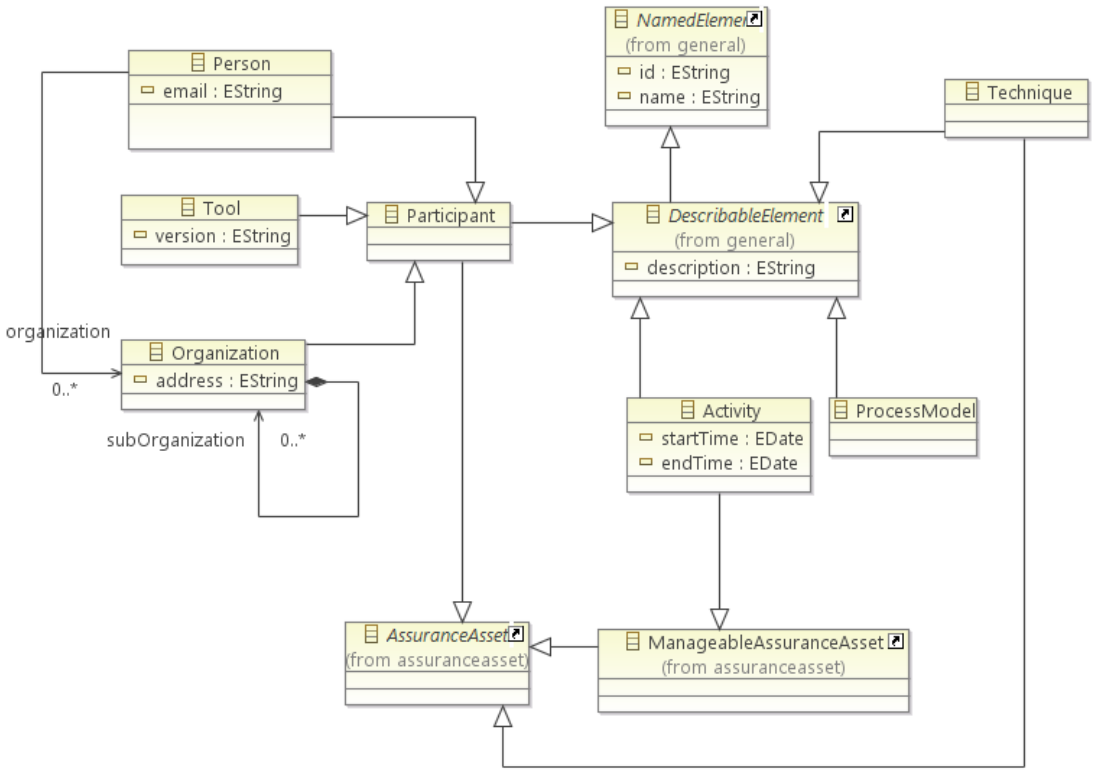


Figure 33. Process Metamodel (Part 2: Inheritance Relationships)

3.4.4.1 ProcessModel

This class maps with ExecutedProcessModel.

Superclass

- *DescribableElement*

Relationships

- ownedActivity: *Activity* [0..*]
  The set of Activities that belongs to the Process Model. This element is an 3.4.2.2 by inheritance.
- ownedParticipant: *Participant* [0..*]
  The set of Participants that belongs to the Process Model. This element is an 3.4.2.2 by inheritance.
- ownedTechnique: *Technique* [0..*]
  The set of Techniques that belongs to the Process Model. This element is an 3.4.2.2 by inheritance.

Semantics

A Process Model specifies the root element of a model representing a set of Process elements. The Process model corresponds to the actual execution of a process with data related to the results to the process execution.

3.4.4.2 Activity

This class maps with ExecutedActivity .

Superclass

- *DescribableElement*
- *ManageableAssuranceAsset*

Attributes

- startTime: *Date*
  The actual date/time when an activity started.
- endTime: *Date*

The actual date/time when an activity finished.

Relationships

- requiredArtefact: *Artefact* [0..*]
  The artefacts necessary for the execution of an activity. These artefacts correspond to the input of the activity.
- producedArtefact: *Artefact* [0..*]
  The artefacts generated or changed in an activity. These artefacts correspond to the output of the activity.
- subActivity: *Activity* [0..*]
  The subactivities executed as part of an Activity which can represent activity instances or any action composing the whole Activity.
- precedingActivity: *Activity* [0..*]
  A set of pointers to the Activities executed before this activity.
- technique: *Technique* [0..*]
  The Technique used in the Activity to generate the produced Artefacts.
- assetEvent: *AssuranceAssetEvent* [0..*]
  A set of Assurance Asset Events generated during the Activity execution.
- ownedRel: *ActivityRel* [0..*]
  The activity relationships owned by an Activity.
- participant: *Participant* [0..*]
  The set of participants being part of the Activity, either executing it or verifying it.
- technique: *Technique* [0..*]
  The set of techniques used in the Activity.

Semantics

An Activity models a unit of work performed in a product lifecycle. An Activity is a specification of an activity already executed.

3.4.4.3 ActivityRel

This class corresponds to existence of a relationship between two activities. This is the main mechanism used to describe the interdependence of activities.

Attributes

- type: *ActivityRelKind*
  The type of relationship between two activities.

Relationships

- target: *Activity* [1]
  The artefact that correspond to the target of an activity relationship.
- source: *Activity* [1]
  The artefact that correspond to the source of an activity relationship.

Semantics

An Activity Relationship models the relationship between two activities. This is the main mechanism for establishing bilateral traceability between activities.

3.4.4.4 Technique

This class corresponds to UsedTechnique.

Superclass

- *DescribableElement*
- *AssuranceAsset*

Relationships

- createdArtefact: *Artefact* [0..*]

The set of *Artefacts* generated using the Technique.

Semantics
A Participant models the parties involved in a product lifecycle.

3.4.4.5 Participant

This class corresponds to Participant.

Superclass
- *DescribableElement*
- *AssuranceAsset*

Relationships
- triggeredAssetEvent: *AssuranceAssetEvent* [0..*]
  The set of AssuranceAssetEvent generated by the Participant.

Semantics
A Participant models the parties involved in a product lifecycle.

3.4.4.6 Person

This class corresponds to individuals that are involved in a product lifecycle.

Superclass
- *Participant*

Attributes
- email: String
  The email address of a person.

Relationships
- organization: *Organization* [0..*]
  The organization for which a person works.

Semantics
A Person models individuals that are involved in a product lifecycle.

3.4.4.7 Tool

This class corresponds to the software tools used in a product lifecycle.

Superclass
- *Participant*

Attributes
- version: String
  The version in use of a tool.

Semantics
A Tool models software tools used in a product lifecycle.

3.4.4.8 Organization

This class corresponds to groups of people (companies, societies, associations, etc.) that are involved in a product lifecycle.

Superclass
- *Participant*

Attributes
- address: String
  The place where an organization is located.

Relationships

- subOrganization: *Organization* [0..*]
  The organization to which an organization belongs.

Semantics
An Organization models groups of people (companies, societies, associations, etc.) that are involved in a product lifecycle.

3.4.4.9 Artefact

See definition in 3.4.3.3

3.4.4.10 ActivityRelKind

See definition in 2.1.2.3.

3.4.4.11 AssuranceAssetEvent

See definition in 3.4.2.5.

3.4.4.12 ManageableAssuranceAsset

See definition in 3.4.2.3.

3.4.4.13 DescribableElement

See definition in 2.1.2.2.

3.4.4.14 NamedElement

See definition in 2.1.2.1.

3.5 Compliance Management Metamodel

3.5.1 Scope and Purpose

See 2.5.1

3.5.2 Implementation Assurance Project Definition

The implementation metamodel has very little differences with the conceptual version. Basically, the links with the UMA as CHESS metamodel don't exist, the plan is a Baseline Model instead of a Standard Model and has a the extra PermissionConfig Metaclass that will be explained in 3.5.2.2.

The class diagram for this Metamodel is presented in the figure below and these differences will be explained in the following subsections.

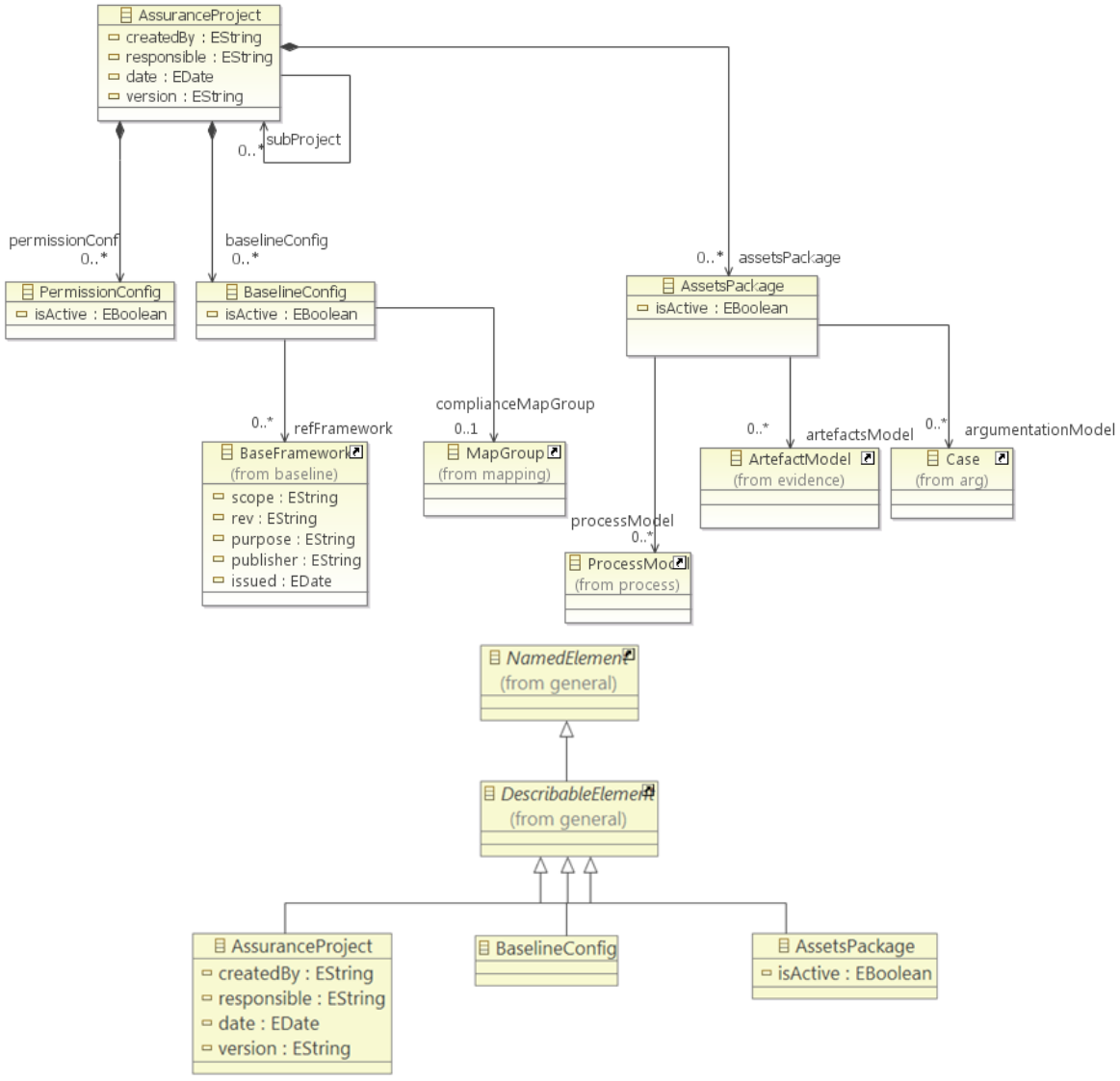


Figure 34. Assurance Project Metamodel

3.5.2.1 AssuranceProject

This class maps with 2.5.2.1.

Superclass

- *DescribableElement*

Attributes

- createdBy: *String*
  The name of the person in charge of creating the Assurance Project.
- responsible: *String*
  The name of the person in charge of managing the life cycle of the Assurance Project.
- date: *Date*
  The date of creation of the Assurance Project.
- version: *String*
  The current version of the Assurance Project.

Relationships

- permissionConf: *PermissionConfig* [0..*]

A reference to PermissionConfig, which is the set of parameters that define the access and permission configuration for the Assurance Project. Only one of the PermissionConfig must be "active" for the Assurance Project.

- baselineConfig: *BaselineConfig* [0..*]
  A reference to baselineConfig, which is what is planned to do or to comply with, in the Assurance Project. Only one of the BaselineConfig must be "active" for the Assurance Project.
- assetsPackage: *AssetsPackage* [0..*]
  A reference to AssetsPackage, which is what has been done in a specific assurance project (project-specific Artefacts models, and Argumentation models, and Process models). Only one of the AssetsPackage must be "active" for the Assurance Project.

Semantics

An Assurance Project models the whole set of elements of a safety assurance project for a system or component, its lifecycle, and any project baseline information that may be shared by the different functional modules.

3.5.2.2 PermissionConfig

This class corresponds to a pointer to a Permission model (not implemented yet in the current metamodel version). The default CDO security model has been implemented on the server side to support profile creation to enable restricted access to the functionality and data.

Attributes

- isActive: *Boolean*
  It indicates if the Permission Configuration is active in the context of the Assurance Project.

Relationships

- none.

Semantics

A Permission Configuration models a pointer to a Permission model (not implemented yet in the current metamodel version). A Permission model will support profile creation to enable restricted access to the functionality and data.

3.5.2.3 BaselineConfig

This class maps with 2.5.2.2.

Superclass

- *DescribableElement*

Attributes

- isActive: *Boolean*
  This flag indicates if the current BaselineConfig is active for its use during the lifecycle of an Assurance Project.

Relationships

- refFramework: *BaseFramework* [0..*]
  The set of BaseFrameworks that are part of the BaselineConfig. E.g., a BaseFramework can correspond to the tailoring of DO-178C (Standard of Sw for Avionics) and another BaseFramework to the tailoring of DO-254 (Standard of Hw for Avionics)
- complianceMapGroup: *MapGroup* [0..1]
  The MapGroup used to refer to a set of Compliance Maps which are valid for the current BaselineConfig.

Semantics

A Baseline Configuration models what is planned to do or comply with, in a specific assurance project. A Baseline Configuration has a set of Baseline Models. Each Baseline Model results from importing (copying) a Reference Framework model and adding information about its Selection in the current project (it

answers to the question: does a given Reference Framework model element apply to the current Assurance Project?).

3.5.2.4 AssetPackage

This class maps with 2.5.2.3.

Superclass

- *DescribableElement*

Attributes

- isActive: *Boolean*
  This flag indicates if the current AssetsPackage is active for its use during the lifecycle of an Assurance Project.

Relationships

- processModel: *ProcessModel* [0..*]
  The set of Process Execution Models which are part of the current AssetsPackage
- artefactsModel: *ArtefactModel* [0..*]
  The set of ArtefactModels which are part of the current AssetsPackage
- argumentationModel: *Case* [0..*]
  The set of Cases (argumentation models) which are part of the current AssetsPackage

Semantics

An Assets Package models what has been done in a specific assurance project. This is a pointer to project-specific Artefacts models, Argumentation models, and Process execution models. The mapping of these three models with Baseline Models is modelled using the concept of Compliance Map.

3.5.2.5 BaseFramework

See definition in 3.5.5.1.

3.5.3 Implementation Process Definition Metamodel

As mentioned above, UMA has been used for process definition. This metamodel is the extension of Conceptual Process Definition Metamodel. We do not provide full meta-class description in this document. For further information on UMA, please refer to [5]. Actually, UMA provides basic access and editing support to the method and process elements stored in a method library. UMA defines the meta-model for how the EPF method content and processes are structured.

In the context of the AMASS project, the integration between process engineering, architecture design and variability management has been implemented. The interested reader may refer to the D6.3 deliverable [7] . The concrete UMA model classes can be grouped into two broad categories:

Method Content

Method content describes roles, the tasks that they perform, the work products produced by the tasks performed and supporting guidance. They can be categorized into logical groups for indexing and display purposes. Method content elements are independent of a development lifecycle. In fact, they are often reused in multiple development lifecycles.

Method content can be sub-divided into the following categories:

- Core Method Content - role, task and work product (artifact, outcome and deliverable).
- Guidance - checklist, concept, example, guideline, estimation considerations, practice, report, reusable asset, roadmap, supporting material, template, term definition, tool mentor and whitepaper.
- Content Category - discipline grouping, discipline domain, work product kind, role set grouping, role set tool and custom category.

The following metamodel diagram shows the organization of the method content classes. They are generated from the uma.ecore file.

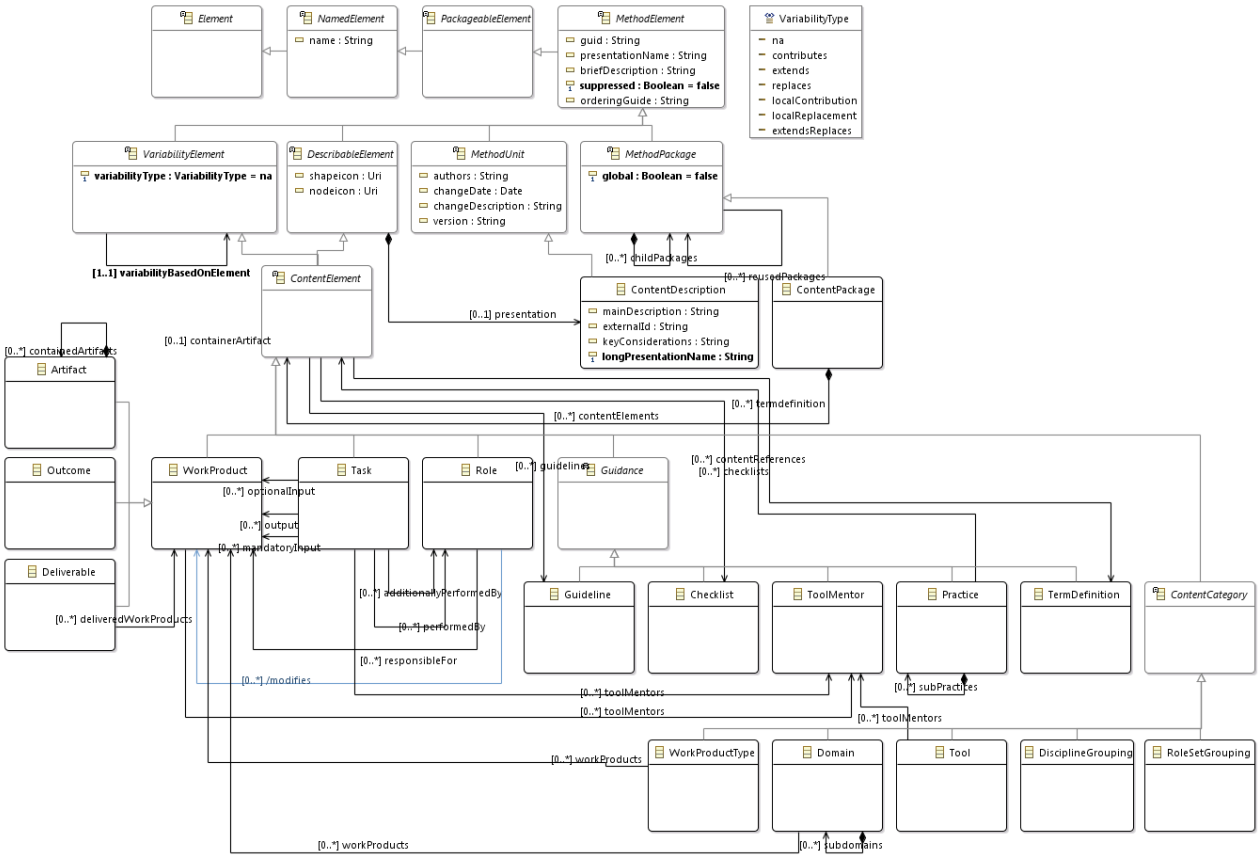


Figure 31. Method Content from the UMA metamodel

Process

Processes describe the development lifecycle. They define sequences of tasks performed by roles and work products produced over time. Processes are typically expressed as workflows or breakdown structures. The sequencing of the tasks within the breakdown structure usually represents different types of development lifecycles, such as waterfall, incremental, and iterative.

The following metamodel diagram shows the organization of the process element classes.

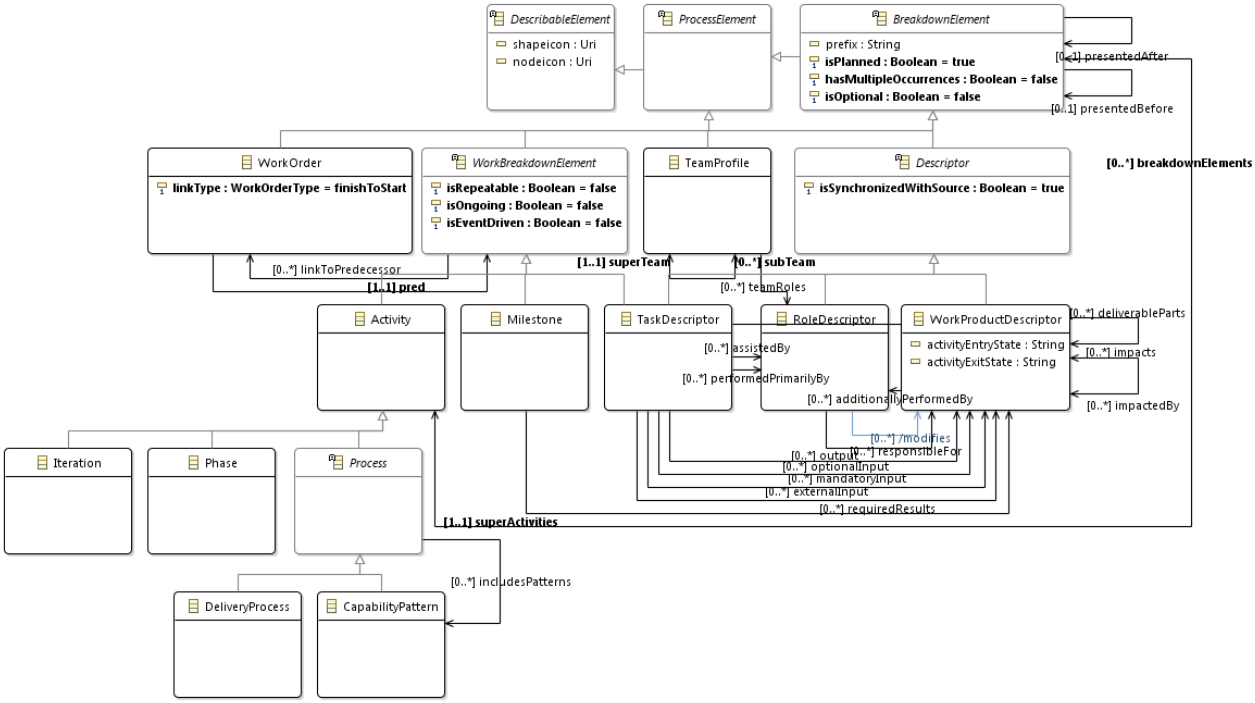


Figure 32. Process from the UMA metamodel

3.5.4 Implementation Standard Definition Metamodel

The Metamodel for the implementation of a Standard is an extension of the conceptual one, and the map can be seen easily because the name of the metaclasses are the same in both except for the RefStandard element that is called RefFramework.

The class diagram for the Reference Assurance Framework Metamodel is shown in Figure 35 and Figure 36 below. In the following subsections, the model elements are defined.

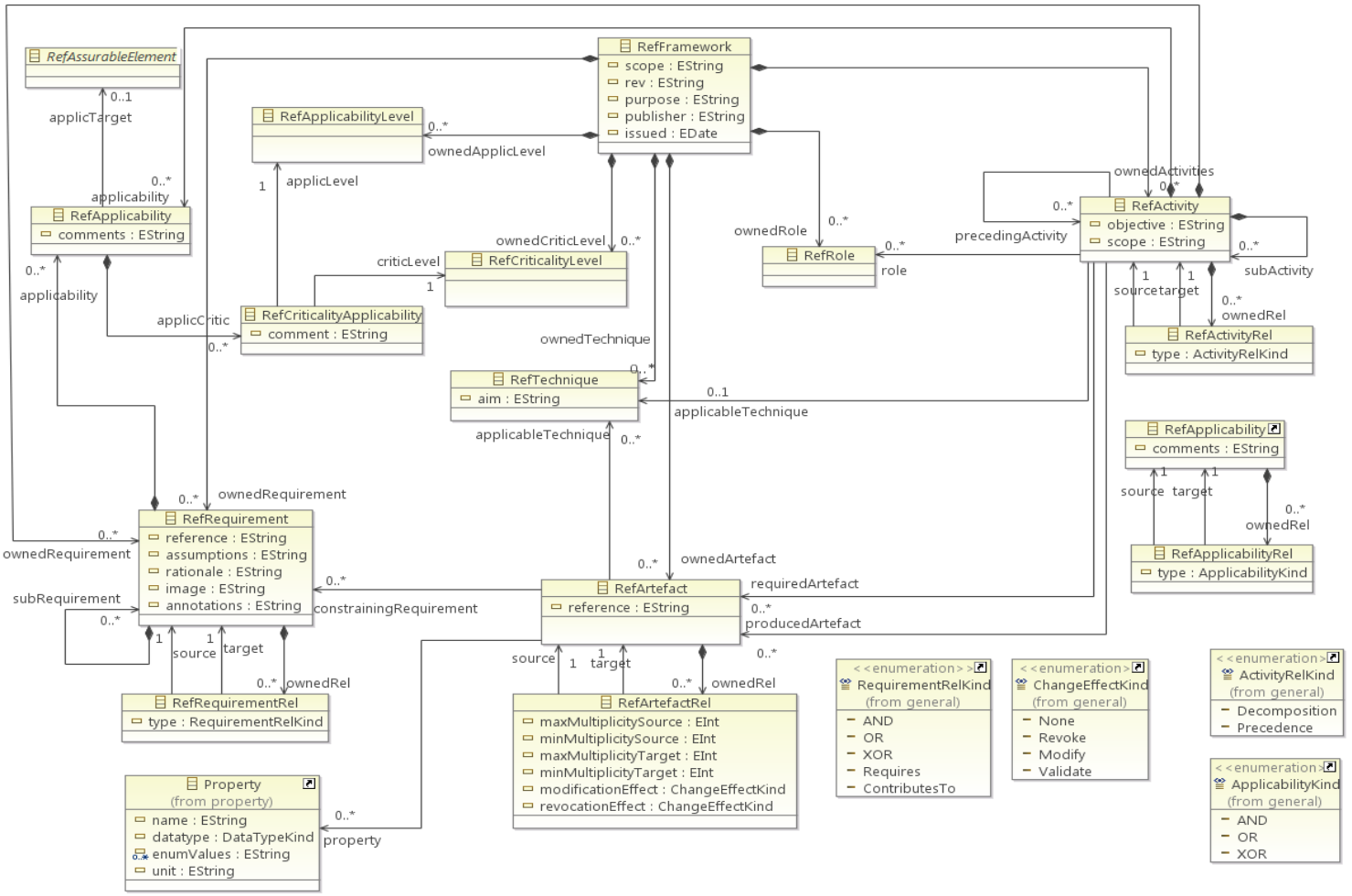


Figure 35. Reference Assurance Framework Metamodel (Part 1: Core Model Elements)

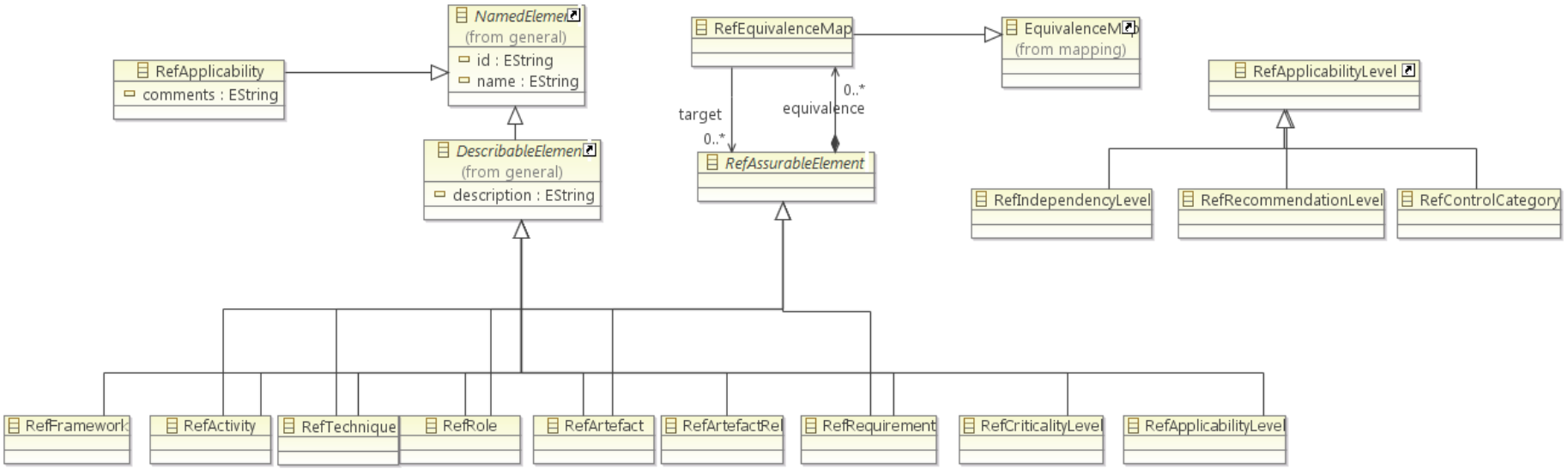


Figure 36. Reference Assurance Framework Metamodel (Part 2: Inheritance Relationships)

3.5.4.1 RefFramework

This class corresponds to a framework to which the lifecycle of a critical system might have to show compliance (for example, a framework based on *IEC61508*).

Superclass

- *DescribableElement*

Attributes

- *scope: String*
  The scope of the reference framework
- *rev: String*
  The revision (version) of the reference framework
- *purpose: String*
  The purpose of the reference framework
- *publisher: String*
  The publisher of the reference framework
- *issued: Date*
  The issue date of the reference framework

Relationships

- ownedRequirement: *RefRequirement* [0..*]
  The (compliance) requirements defined in a reference assurance framework.
- ownedActivities: *RefActivity* [0..*]
  The reference activities defined in a reference assurance framework.
- ownedRole: *RefRole* [0..*]
  The roles defined in a reference assurance framework.
- ownedArtefact: *RefArtefact* [0..*]
  The reference artefacts defined in a reference assurance framework.
- ownedTechnique: *RefTechnique* [0..*]
  The references techniques defined in a reference assurance framework.
- ownedCriticlevel: *RefCriticalityLevel* [0..*]
  The criticality levels defined in a reference assurance framework.
- ownedApplicLevel: *RefApplicabilityLevel* [0..*]
  The applicability levels defined in a reference assurance framework.

Semantics

A Reference Assurance Framework is the main container to model concepts against which the safety and system engineering aspects of a given system are developed and assessed, for example, safety standards such as IEC 61508, ISO 26262, DO-178C, EN 50126, company standards and best practice documentation (e.g., the Alstom, Thales or Fiat process to develop safety-critical systems), as well as documents which have the de facto status of standards, such as, for example, the Aerospace Recommended Practice (ARP) documents (e.g. ARP 4754 *Certification Considerations for Highly-Integrated or Complex Aircraft Systems*) [19].

3.5.4.2 RefActivity

This class corresponds to the units of behaviour that a reference assurance framework defines for the system lifecycle and that must be executed to demonstrate compliance.

Superclass

- *DescribableElement*
- *RefAssurableElement*

Attributes

- *objective: String*

The objective of the reference activity

- *scope: String*
  The scope of the reference activity

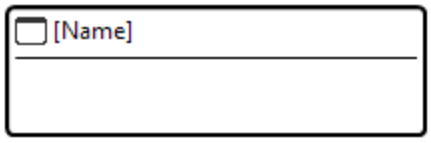
Relationships

- ownedRequirement: *RefRequirement* [0..*]
  The requirements that must be fulfilled after (including during) the execution of a reference activity.
- role: *RefRole* [0..*]
  The roles responsible for the realisation of a reference activity.
- requiredArtefact: *RefArtefact* [0..*]
  The reference artefacts necessary for the execution a reference activity. These reference artefacts correspond to the input of the reference activity.
- producedArtefact: *RefArtefact* [0..*]
  The reference artefacts generated or changed in a reference activity. These reference artefacts correspond to the output of the reference activity.
- ApplicableTechnique: *RefTechnique* [0..*]
  The reference techniques used for the execution of a reference activity.
- subActivity: *RefActivity* [0..*]
  The more fine-grained reference activity which is part of the reference activity.
- precedingActivity: *RefActivity* [0..*]
  The preceding reference activity that must be executed before the reference activity.
- applicability: *RefApplicability* [0..*]
  The reference applicability specification of a reference activity.
- ownedRel: *RefActivityRel* [0..*]
  The activity relationship owned by the reference activity.

Semantics

A Reference Activity is the first-class modeling entity of process specifications. It defines a phase, activity, tasks, or action, depending on the activity granularity level, defined in a standard or company process. It also relates a number of concepts such as the artefact required and produced, roles involved in activities, techniques used and levels of applicability according to criticality levels.

Graphical Notation



Preceding Activity (Activity2 precedes Activity1):

3.5.4.3 RefActivityRel

This class corresponds to the existence of a relationship between two reference activities.

Attributes

- type: *ActivityRelKind*
  The type of relationship between two reference activities.

Relationships

- source: *RefActivity* [1]
  The reference activity that corresponds to the source of a reference activity relationship.
- Target: *RefActivity* [1]

The reference activity that corresponds to the target of a reference activity relationship.

Semantics

A Reference Activity Relationship models different kinds of relationships between two reference activities. The semantics of the relationships are defined by the ActivityRelKind enumeration.

3.5.4.4 RefRole

This class corresponds to the types of agents that execute a reference activity.

Superclass

- *DescribableElement*
- *RefAssurableElement*

Semantics

A Reference Role models any agent involved in the execution of a reference activity.

Graphical Notation

3.5.4.5 RefArtefact

This class corresponds to the types of units of data that a Reference Assurance Framework defines and that must be created and maintained during system lifecycle to demonstrate compliance. Reference artefacts are materialised in assurance projects by means of (concrete) artefacts. This means that these artefacts have the same or a similar structure (syntax) and/or purpose (semantics). Please note that an artefact is not necessarily to be interpreted as a document. An artefact should be an atomic and coherent piece of information. Documents can therefore be conceived as being “practical containers” for many artefacts, which can be read sequentially (but need not necessarily be). An electronic project repository, for example, might allow for navigation and search over artefacts without the need for traditional (i.e. printed) documents. We refer to this as a “model-centric” approach.

Superclass

- *DescribableElement*
- *RefAssurableElement*

Attributes

- reference: *String*
  The description of any reference to a given reference artefact.

Relationships

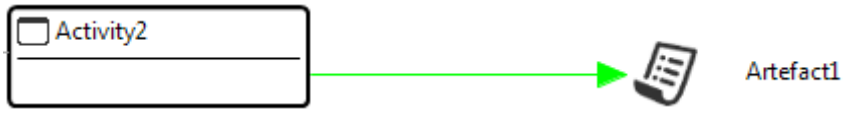
- ownedRel: *RefArtefactRel* [0..*]
  The reference artefact relationships owned by a reference artefact.
- property: *Property* [0..*]
  The reference artefact properties corresponding to the actual artefact can have property values.
- constrainingRequirement: *RefRequirement* [0..*]
  The requirements at which a reference artefact is targeted.
- applicableTechnique: *RefTechnique* [0..*]
  The techniques used to create a reference artefact.

Semantics

The Reference Artefact models units of data that a reference assurance framework defines and that must be created and maintained during system lifecycle to demonstrate compliance. Reference artefacts are materialised in assurance projects by means of (concrete) artefacts. This means that these artefacts have the same or a similar structure (syntax) and/or purpose (semantics).

Graphical Notation

Produced Artefact:



Required Artefact:

3.5.4.6 RefArtefactRel

This class corresponds to the existence of a relationship between two reference artefacts. A reference artefact relationship is materialised by relating two artefacts of an assurance project (i.e., by means of an artefact relationship), and characterizes those artefact relationships that have the same or similar structure (syntax) and/or purpose (semantics).

Superclass

- *DescribableElement*

Attributes

- maxMultiplicitySource: Int
  The maximum number of times that an artefact that materialises the source of a reference artefact relationship can be used as the source of artefact relationships that materialise the reference artefact relationship.
- minMutiplicitySource: Int
  The minimum number of times that an artefact that materialises the source of a reference artefact relationship must be used as the source of artefact relationships that materialise the reference artefact relationship.
- maxMultiplicityTarget: Int
  The maximum number of times that an artefact that materialises the target of a reference artefact relationship can be used as the target of artefact relationships that materialise the reference artefact relationship.
- minMultiplicityTarget: Int
  The minimum number of times that an artefact that materialises the target of a reference artefact relationship must be used as the target of artefact relationships that materialise the reference artefact relationship.
- modificationEffect: *ChangeEffectKind*
  The effect that the modification (or deletion) of an artefact that materialises the target of a reference artefact relationship has on the artefact that materialises the source of the reference artefact relationship.
- revocationEffect: *ChangeEffectKind*
  The effect that the revocation of an artefact that materialises the target of a reference artefact relationship has on the artefact that materialises the source of the reference artefact relationship.

Relationships

- source: *RefArtefact* [1]
  The reference artefact that corresponds to the source of a reference artefact relationship.
- target: *RefArtefact* [1]
  The reference artefact that corresponds to the target of a reference artefact relationship.

Semantics
The Reference Artefact Relationship models a relationship between two reference artefacts. A reference artefact relationship is materialised by relating two artefacts of an assurance project (i.e., by means of an artefact relationship), and characterizes those artefact relationships that have the same or similar structure (syntax) and/or purpose (semantics)

3.5.4.7 RefTechnique

This class corresponds to specific ways to create a reference artefact.

Superclass
- *DescribableElement*
- *RefAssurableElement*

Attributes
- aim: String
  The purpose of a reference technique.

Relationships
- none

Semantics
The Reference Technique models a method used during the system development lifecycle to create an artefact.

3.5.4.8 RefRequirement

This class corresponds to the criteria (e.g., objectives) that a reference assurance framework defines (or prescribes) to comply with it.

Superclass
- *DescribableElement*
- *RefAssurableElement*

Attributes
- reference: String
  The reference of the requirement in the reference framework documents.
- assumptions: String
  The statements considered as preconditions to meet the reference requirement.
- rationale: String
  Any rationale to justify the need to meet the reference requirement.
- image: String
  A placeholder for an image capturing the reference requirement description from the reference framework documents.
- annotations: *String*
  Any complementary annotation clarifying the means to meet the requirement.

Relationships
- subRequirement: *RefRequirement* [0..*]
  A more fine-grained reference requirement of which this reference requirement is composed.
- ownedRel: *RefRequirementRel* [0..*]
  The reference requirement relationships owned by a reference requirement.
- applicability: *RefApplicability* [0..*]
  The reference applicability tuple (composed of applicability level, criticality level and assurable element – such as a technique, a requirement or an activity) of which a reference requirement is composed.

Semantics
The Reference Requirement models the criteria (e.g., objectives) that a reference assurance framework defines (or prescribes) to comply with it.

3.5.4.9 RefRequirementRel

This class corresponds to the existence of a relationship between two requirements.

Attributes
- type: *RequirementRelKind*
  The kind of a requirements relationship.

Relationships
- source: *RefRequirement* [1]
  The reference requirement that corresponds to the source of a reference requirement relationship.
- Target: *RefRequirement* [1]
  The reference requirement that corresponds to the target of a reference requirement relationship.

Semantics
A Reference Requirement Relationship models different kinds of relationships between two reference requirements. The semantics of the relationships are defined by the RequirementRelKind enumeration.

3.5.4.10 RefCriticalityLevel

This class corresponds to the categories of criticality that a reference assurance framework defines and that indicate the relative level of risk reduction being provided (e.g., *SIL 1, 2, 3, and 4* for IEC61508).

Superclass
- *DescribableElement*

Semantics
This Reference Criticality Level models the categories of criticality that a reference assurance framework defines and that indicate the relative level of risk reduction that needs to be provided (e.g., *SIL 1, 2, 3, and 4* for IEC61508).

3.5.4.11 RefApplicabilityLevel

This class corresponds to the categories of applicability that a reference assurance framework defines (e.g., a given technique can be *mandated* in EN50128).

Superclass
- *DescribableElement*

Semantics
This Reference Applicability Level models the categories of applicability that a reference assurance framework defines (e.g., a given technique can be *mandated* in EN50128).

3.5.4.12 RefIndependencyLevel

This class corresponds to the kind of categories of applicability related to the independency required to perform an activity or to achieve and compliance objective that a reference assurance framework defines (e.g., the level of independence of the person performing a verification activity *mandated* in DO-178C).

Superclass
- *RefApplicabilitylevel*

Semantics
This Reference Independency Level models the kind of categories of applicability related to the independency required to perform an activity or to achieve and compliance objective that a reference assurance framework defines (e.g., the level of independence of the person performing a verification activity *mandated* in DO-178C).

3.5.4.13 RefRecommendationLevel

This class corresponds to the kind of categories of applicability related to the level of recommendation of a given activity, artefact or compliance requirement that a reference assurance framework defines (e.g., the degree of recommendation to use the methods that ISO 26262 assigns to each ASIL within a conformity requirement).

Superclass

- *RefApplicabilitylevel*

Semantics

This Reference Recommendation Level models the kind of categories of applicability related to the level of recommendation of a given activity, artefact or compliance requirement that a reference assurance framework defines (e.g., the degree of recommendation to use the methods that ISO 26262 assigns to each ASIL within a conformity requirement).

3.5.4.14 RefControlCategory

This class corresponds to the kind of categories of applicability related to the data control category associated to the configuration management controls placed on the data. (e.g., the CC1 and CC2 control categories defined in DO-178C).

Superclass

- *RefApplicabilitylevel*

Semantics

This Reference Control Category models the kind of categories of applicability related to the data control category associated to the configuration management controls placed on the data. (e.g., the CC1 and CC2 control categories defined in DO-178C).

3.5.4.15 RefCriticalityApplicability

This class corresponds to the assignation, in a reference assurance framework, of an applicability level for a given criticality level to a reference applicability.

Attributes

- comment: *String*
  The comments that are embedded in applicability tables, which can imply constraints on the applicability specification.

Relationships

- applicLevel: *RefApplicabilityLevel* [1]
  The applicability levels of the criticality applicability.
- criticLevel: *RefCriticalityLevel* [1]
  The criticality level of the criticality applicability.

Semantics

The Reference Criticality Applicability models the pair of an applicability level for a given criticality level to a RefApplicability.

3.5.4.16 RefApplicability

This class corresponds to the reference applicability tuple (composed of applicability level, criticality level and assurable element – such as a technique, a requirement or an activity) a reference requirement is composed of.

Superclass

- *NamedElement*

Attributes

- comments: *String*

The comments that are embedded in applicability tables, which can imply constraints on the applicability specification.

Relationships

- applicTarget: *BaseAssurableElement* [0..1]
  The assurable element – such as a technique, a requirement or an activity, to which a reference applicability applies to.
- applicCritic: *RefCriticalityApplicability* [0..*]
  The pair of criticality and applicability levels applied to the targeted assurable element.

Semantics

This Reference Applicability models the reference applicability tuple (composed of applicability level, criticality level and assurable element – such as a technique, a requirement or an activity) of which a reference requirement is composed.

3.5.4.17 RefApplicabilityRel

This class corresponds to the existence of a relationship between two reference applicability specifications.

Attributes

- type: *ApplicabilityKind*
  The kind of an applicability relationship.

Relationships

- source: *RefApplicability* [1]
  The reference applicability that corresponds to the source of a reference applicability relationship.
- Target: *RefApplicability* [1]
  The reference applicability that corresponds to the target of a reference applicability relationship.

Semantics

A Reference Applicability Relationship models different kinds of relationships between two reference applicability specifications. The semantics of the relationships are defined by the ApplicabilityKind enumeration.

3.5.4.18 RefAssurableElement (Abstract)

This class factorises various model elements used to specify assurance concepts, including reference activities, techniques, artefacts, roles, and requirements.

Relationships

- equivalence: *RefEquivalenceMap* [0..*]
  The equivalence map to which an assurable element is mapped.

Semantics

The Reference Assurable Element model elements used to specify assurance concepts, including reference activities, techniques, artefacts, and requirements.

3.5.4.19 RefEquivalenceMap

This class specifies a single mapping between two or more assurable elements.

Relationships

- target: *BaseAssurableElement* [0..*]
  The reference assurable element which a map is targeting.

Semantics

The Reference Equivalence Map models a single mapping between two or more assurable elements.

3.5.4.20 ActivityRelKind (enumeration)

See definition in 2.1.2.3.

3.5.4.21 ChangeEffectKind (enumeration)

See definition in 2.1.2.4.

3.5.4.22 RequirementRelKind (enumeration)

See definition in 2.1.2.5.

3.5.4.23 ApplicabilityKind (enumeration)

See definition in 2.1.2.6.

3.5.4.24 Property

See definition in 2.1.3.2.

3.5.4.25 EquivalenceMap

See definition in 2.5.6.4.

3.5.4.26 DescribableElement

See definition in 2.1.2.2.

3.5.4.27 NamedElement

See definition in 2.1.2.1.

3.5.5 Implementation Baseline Definition Metamodel

The Baseline Definition Metamodel captures what is planned to be done or to be complied with a concrete standard, in a specific assurance project.

This metamodel is a copy of the standard metamodel renaming elements from “Refxxx” to “Basexx”, the addition of the BaselineElement class to indicate if the element of the standard is part of the plan or not and the justification, and the link with the compliance mapping metamodel to allow “mark” parts of the plan things as “done”.

The metamodel is shown in the figures below and only the differences will be explained in the following subsections.

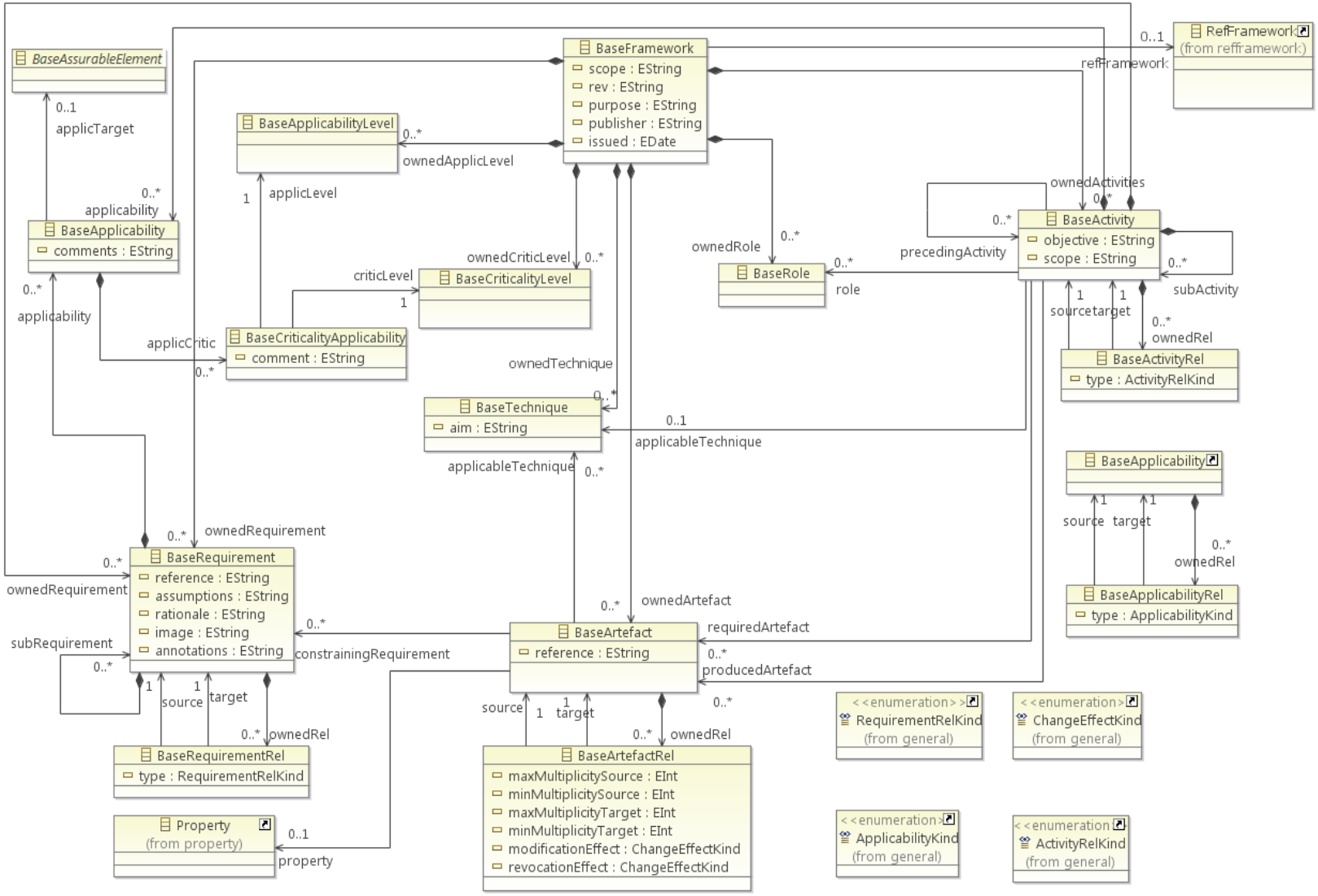


Figure 37. Baseline Definition metamodel (Part 1: Core Model Elements)

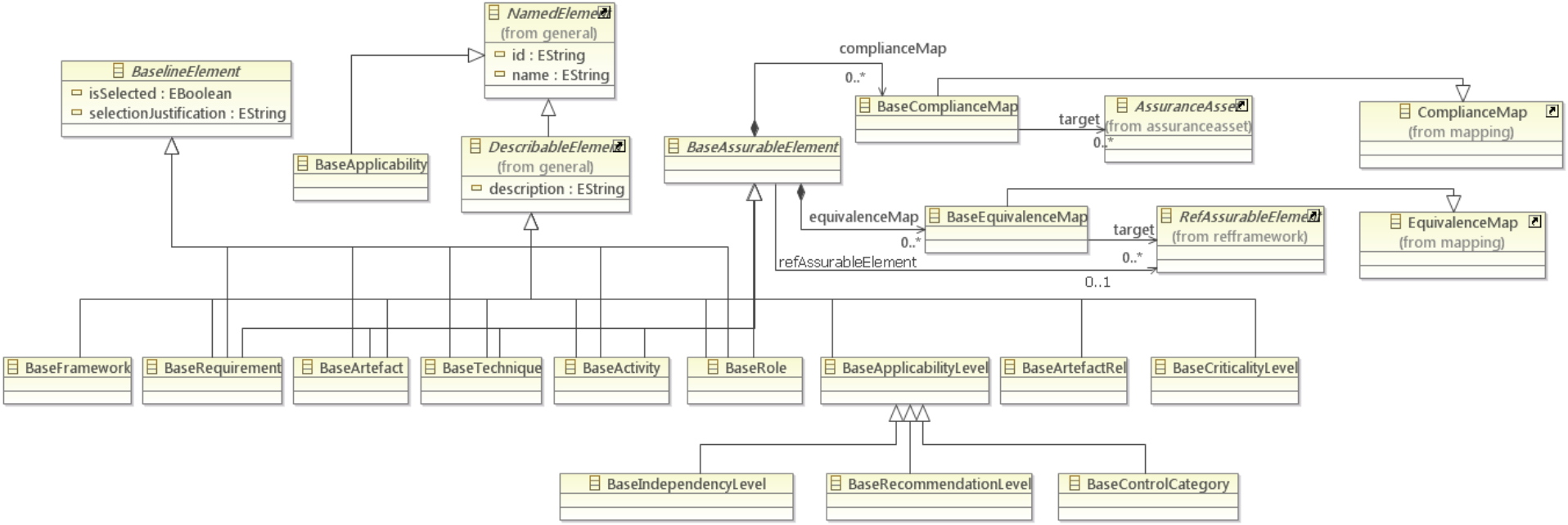


Figure 38. Baseline Definition metamodel (Part 2: Inheritance Relationship)

3.5.5.1 BaseFramework

This class corresponds to 3.5.4.1 class but it has an extra relation:

- refFramework: *RefFramework* [0..1].
  The RefFramework used to create the BaseFramework.

3.5.5.2 BaseActivity

This class corresponds to 3.5.4.2.

3.5.5.3 BaseActivityRel

This class corresponds to 3.5.4.3.

3.5.5.4 BaseRole

This class corresponds to 3.5.4.4

3.5.5.5 BaseArtefact

This class corresponds to 3.5.4.5

3.5.5.6 BaseArtefactRel

This class corresponds to 3.5.4.6

3.5.5.7 BaseTechnique

This class corresponds to 3.5.4.7.

3.5.5.8 BaseRequirement

This class corresponds to 3.5.4.8.

3.5.5.9 BaseRequirementRel

This class corresponds to 3.5.4.9.

3.5.5.10 BaseCriticalityLevel

This class corresponds to 3.5.4.10.

3.5.5.11 BaseApplicabilityLevel

This class corresponds to 3.5.4.11.

3.5.5.12 BaseIndependencyLevel

This class corresponds to 3.5.4.12.

3.5.5.13 BaseRecommendationLevel

This class corresponds to 3.5.4.13.

3.5.5.14 BaseControlCategory

This class corresponds to 3.5.4.14.

3.5.5.15 BaseCriticalityApplicability

This class corresponds to 3.5.4.15.

3.5.5.16 BaseApplicability

This class corresponds to 3.5.4.16.

3.5.5.17 BaseApplicabilityRel

This class corresponds to 3.5.4.17.

3.5.5.18 BaselineElement (abstract)

This class factorises various model elements used to specify baseline flags to indicate if the model element has been selected to be active in the concrete assurance project, including base activities, techniques, artefacts, roles, and requirements.

Attributes

- isSelected: *Boolean*
  The flag to indicate if the model element has been selected to be active in the associated assurance project.
- selectionJustification: *String*
  A string to specify why a model element has been or not selected to be active in the associated assurance project.

Semantics

A Baseline Element models any model element that can have a baseline flags to indicate if the model element has been selected to be active in the concrete assurance project, including base activities, techniques, artefacts, roles, and requirements.

3.5.5.19 BaseAssurableElement (Abstract)

This class factorises various model elements used to specify assurance concepts, including base activities, techniques, artefacts, roles, and requirements in a concrete assurance project.

Relationships

- equivalenceMap: *BaseEquivalenceMap* [0..*]
  The equivalence map to which an assurable element is mapped.
- complianceMap: *BaseComplianceMap* [0..*]
  The compliance map assurable map to which an assurable element is mapped.
- refAssurableElement. The link to the original element from the standard.

Semantics

The Base Assurable Element model elements, used to specify assurance concepts, including base activities, techniques, artefacts, and requirements.

3.5.5.20 BaseEquivalenceMap

This class corresponds to 3.5.4.19.

3.5.5.21 BaseComplianceMap

This class specifies a single compliance mapping between an assurable element and an assurance asset.

Superclass

- *ComplianceMap*

Relationships

- target: *AssuranceAsset* [0..*]

The assurance asset element which a map is targeting.

Semantics

The Base Compliance Map models a single compliance mapping between an assurable element and an assurance asset.

3.5.5.22 ActivityRelKind (enumeration)

See definition in 2.1.2.3.

3.5.5.23 ChangeEffectKind (enumeration)

See definition in 2.1.2.4.

3.5.5.24 RequirementRelKind (enumeration)

See definition in 2.1.2.5.

3.5.5.25 ApplicabilityKind (enumeration)

See definition in 2.1.2.6.

3.5.5.26 Property

See definition in 2.1.3.2.

3.5.5.27 DescribableElement

See definition in 2.1.2.2.

3.5.5.28 NamedElement

See definition in 2.1.2.1.

3.5.5.29 RefAssurableElement

See definition in 3.5.4.18.

3.5.5.30 AssuranceAsset

See definition in 3.4.2.2.

3.5.5.31 ComplianceMap

See definition in 2.5.6.5.

3.5.5.32 EquivalenceMap

See definition in 2.5.6.4.

3.5.6 Implementation Vocabulary Metamodel

The implementation metamodel is a simplification of the conceptual metamodel and is described in the figure below and the subsequent subsections.

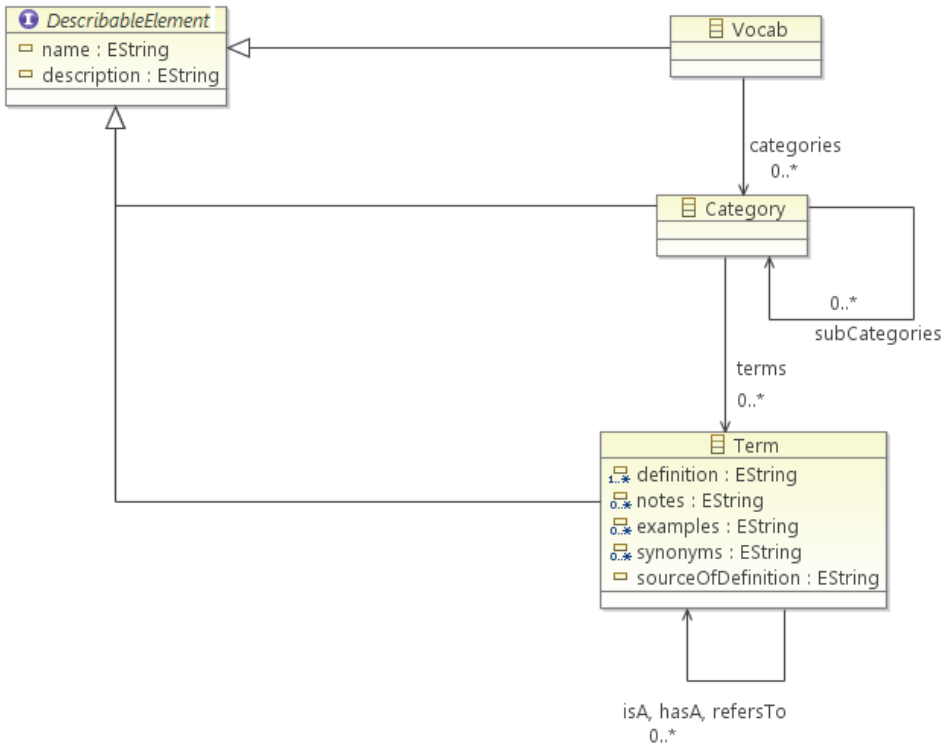


Figure 39. Vocabulary Metamodel

3.5.6.1 Term

The Term class defines the language primitives (words and expressions) which are used to construct the names and definitions of entities in the CACM Thesaurus and to populate the models to be developed at Levels 1, 1b and 2. Terms are the linguistic labels used to represent/reify instances of Concepts.

Superclass

- *DescribableElement*

Attributes

- definition: string - prose definition of the concept, expressed in phrases.
- notes: string – prose information relating to the concept, which are not directly incorporated in the definition
- examples: string – prose descriptions of how the concept is used in an assurance project
- synonyms: string – list of other Terms which have identical definitions and applications to the term being modelled
- sourceOfDefinition: string – an indication of the source of the definition and the term (i.e. a particular standard or wordlist). It is important to give precise location information here, to mitigate the issues of inconsistent language within the source documents.

Relationships

- isA [0..*] – a Term is an instance of a broader concept, captured by another Term (cf. Definition of hypernymy above)
- hasA [0..*] – A Term represents a broader concept which comprises several other concepts represented by Terms (cf. Definition of holonymy above)
- refersTo [0..*] – A Term refers to another Term (for example in the definition attribute)

3.5.6.2 Category

The CACM Thesaurus will be structured according to the grouping of Terms by the concepts they represent (and the relationships between these concepts). A term may belong to more than one Category, depending on its usage. See discussion of ConceptType above.

Superclass

- *Describable Element*

Relationships

- subCategories [0..*] – a category may optionally be further subdivided into subcategories.
- terms [0..*] - a Term may be a member of a Category

3.5.6.3 Vocab

This class defines the set of Terms which is used in a given Assurance Project or Defined Standard. The vocabulary includes single words and phrases and their definitions (modelled in the CACM as an attribute of Term) and defines the scope of the conceptual categories by which it is structured.

Superclass

- Describable Element

Relationships

categories [0..*] – a vocab may be structured according to Categories (see discussion of ConceptType above).

3.5.6.4 DescribableElement

See definition in 2.1.2.2.

3.5.7 Implementation Mapping Definition Metamodel

The implementation and the conceptual model for Mapping Definition are the same.

Abbreviations

AADL	Architecture Analysis & Design Language
ALF	Action Language for Foundational UML
API	Application Programming Interface
ARM	Argumentation Metamodel
ARP	Aerospace Recommended Practice
ARTA	AMASS Reference Tool Architecture
ASIL	Automotive Safety Integrity Level
BMM	Business Motivation Model
BPMN	Business Process Model and Notation
BVR	Base Variability Resolution
CACM	Common Assurance & Certification Metamodel
CCL	Common Certification Language
CDO	Connected Data Objects
CHESSML	CHESS Modelling Language
CMMA	Component MetaModel supporting Architecture-driven assurance
CPS	Cyber-Physical Systems
CSV	Comma Separated Value
CVS	Concurrent Version System
DAF	Dependability Assurance Framework for Safety-Sensitive Consumer Devices
DAL	Development Assurance Level
DSL	Domain Specific Language
EMF	Eclipse Modelling Framework
EPF	Eclipse Process Framework)
FAA	Federal Aviation Administration
FMEA	Failure mode and effects analysis
FTA	Fault Tree Analysis
fUML	Foundational Subset for Executable UML Models
GMF	Graphical Modelling Framework
GSN	Goal Structuring Notation
GUI	Graphical User Interface
HMI	Human-Machine Interface
HW	Hardware
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
MBD	Model-based design
MDSD	Model-driven software development
OCL	Object Constraint Language
OMG	Object Management Group
OPENCOSS	Open Platform for EvolutioNary Certification Of Safety-critical Systems
OSLC	Open Services for Lifecycle Collaboration
PASRA	Preliminary Aircraft Security Risk Assessment
PSCS	Precise Semantics of UML Composite Structures
RMC	Rational Method Composer
RSA	Rational Software Architect
RTDS	Real Time Developer Studio
SACM	Structured Assurance Case Metamodel
SIL	Safety Integrity Level
SEooCMM	Safety Element out-of-context Metamodel
SMM	Structured Metrics Meta-model

SOAML	Service Oriented Architecture Modelling Language
SPEM	Software & Systems Process Engineering Metamodel
STO	Scientific and Technical Objective
SVN	Subversion
SW	SoftWare
SysML	Systems Modelling Language
TRL	Technology Readiness Level
UMA	Unified Method Architecture
UML	Unified Modelling Language
UML-RT	UML for Real-Time
UTP	UML Testing Profile
WP	Work Package

References

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

[2] SafeCer https://artemis-ia.eu/project/40-nsafecer.html

[3] AMASS D2.2 - AMASS Reference Architecture (a), November 2016

[4] AMASS D2.4 - AMASS Reference Architecture (c), June 2018

[5] AMASS D2.9 - AMASS Platform Validation, January 2019

[6] AMASS D3.3 - Design of the AMASS tools and methods for architecture-driven assurance (b), March 2018

[7] AMASS D6.3 - Design of the AMASS tools and methods for cross/intra-domain reuse (b), July 2018

[8] SafeCer Deliverable D132.2, Generic component meta-model v1.0, 2014-12-19

[9] I. Sljivo, B. Gallina, J. Carlson, H. Hansson, and S. Puri, 'A Method to Generate Reusable Safety Case Fragments from Compositional Safety Analysis', in Software Reuse for Dynamic Systems in the Cloud and Beyond, Springer, 2014, pp. 253–268.

[10] OMG: Structured Assurance Case Metamodel (SACM). http://www.omg.org/spec/SACM/ (accessed 2016-10-31)

[11] OMG, "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

[12] E. Foundation. Eclipse process framework (epf) composer 1.0 architecture overview. http://www.eclipse.org/epf/composer architecture/. Accessed: 2016-08-29.

[13] OPENCOSS: Deliverable 4.4 - Common Certification Language: Conceptual Model. http://www.opencoss-project.eu/sites/default/files/D4.4_v1.5_FINAL.pdf (accessed 2016-10-31)

[14] OMG: Dependability Assurance Framework for Safety-Sensitive Consumer Devices (DAF), version 1.0. http://www.omg.org/spec/DAF/ (accessed 2016-10-31)

[15] "IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems," International Electrotechnical Commission, Geneva, Switzerland, Standard, 2010.

[16] "ISO 26262: Road vehicles Functional safety," International Organization for Standardization, Geneva, Switzerland, Standard, 2011.

[17] "DO-178C: Software Considerations in Airborne Systems and Equipment Certification," Radio Technical Commission for Aeronautics, Washington, USA, Standard, Jan. 2012.

[18] "EN 50126: Railway applications. The specification and demonstration of reliability, availability, maintainability and safety (RAMS). Basic requirements and generic process," European Committee for Electrotechnical Standarization, Brussels, Belgium, Standard, 1999.

[19] "ARP4754A: Guidelines for Development of Civil Aircraft and Systems," SAE International, Brussels, Belgium, Standard, 2010. http://standards.sae.org/arp4754/

[20] CHESS modelling language, https://www.polarsys.org/chess/publis/CHESSMLprofile.pdf

[21] Object Modelling Group, Semantics of Business Vocabulary and Business Rules (SBVR), 2008

[22] SafeCer Deliverable D132.2, Generic component meta-model v1.0, 2014-12-19

[23] I. Sljivo, B. Gallina, J. Carlson, H. Hansson, and S. Puri, 'A Method to Generate Reusable Safety Case Fragments from Compositional Safety Analysis', in Software Reuse for Dynamic Systems in the Cloud and Beyond, Springer, 2014, pp. 253–268.

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

[25] VARIES: D4.2 BVR - The language https://github.com/SINTEF-9012/bvr/blob/master/docs/VARIES_D4.2_v01_PP_FINAL.pdf