ECSEL Research and Innovation actions (RIA)

AMASS

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

AMASS Reference Architecture (c)
D2.4

Work Package:	WP2: Reference Architecture and Integration
Dissemination level:	PU = Public
Status:	Final
Date:	4th June 2018
Responsible partner:	Garazi Juez | Alejandra Ruiz (TECNALIA)
Contact information:	garazi.juez@tecnalia.com | alejandra.ruiz@tecnalia.com
Document reference:	AMASS_D2.4_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
Alejandra Ruiz, Estibaliz Amparan, Garazi Juez, Angel López, Cristina Martinez	TECNALIA Research & Innovation (TEC)
Jose Luis de la Vara, Jose María Álvarez, Eugenio Parra, Pablo Sánchez, Juan Llorens	Universidad Carlos III de Madrid (UC3)
Luis Alonso, Borja López	The REUSE Company (TRC)
Stefano Puri	Intecs (INT)
Barbara Gallina, Irfan Sljivo, Muhammad Atif Javed, Faiz UL Muram	Maelardalen Hoegskola (MDH)
Thomas Gruber, Dejan Ničković, Niveditha Manjunath	AIT Austrian Institute of Technology (AIT)
Stefano Tonetta, Alberto Debiasi	Fondazione Bruno Kessler (FBK)
Tomáš Kratochvíla	Honeywell (HON)

Reviewers

Names	Organisation
Jose Luis de la Vara (Peer-reviewer)	Universidad Carlos III de Madrid (UC3)
Marc Sango (Peer-reviewer)	Alliance pour les Technologies de l' Informatique (A4T)
Cristina Martinez (Quality Review)	Tecnalia Research & Innovation (TEC)

TABLE OF CONTENTS

Executive Summary **9**

1. Introduction **10**
1.1 Context 10
1.2 Purpose (*) 11
1.3 Technical Context and Objectives 12
1.4 Relation to other AMASS Tasks (*) 12

2. AMASS Reference Tool Architecture (ARTA) **13**
2.1 Goals and Philosophy 13
2.2 Tool Architecture Outline (*) 13
2.3 Stakeholders 19
2.4 AMASS Tool Platform (*) 20

3. Architectural Views **22**
3.1 Logical View 23
3.1.1 Infrastructure Use Cases 24
3.1.2 Architecture-driven Assurance Uses Cases (*) 30
3.1.3 Multi-concern Assurance Uses Cases (*) 46
3.1.4 Seamless Interoperability Use Cases 56
3.1.5 Cross-Intra Domain Reuse Use Cases 61
3.2 Structural View (*) 68
3.2.1 Infrastructure Module 69
3.2.2 Application Module 71
3.3 Interactional View 77
3.3.1 Interaction Assurance Case Development Scenario 77
3.3.2 Interaction Modular Argumentation/Architecture Development Scenario 78
3.3.3 Interaction Process Based Argumentation Scenario Development 79
3.3.4 Interaction Multi-Concern Co-Analysis/Assessment Scenario 79
3.3.5 Interaction Development Cross/Intra-Domain Reuse of Base Models 80
3.3.6 Interaction User Access and Concurrent Evidence Management 81
3.3.7 Interaction Evidence Information Exchange and Traceability 82
3.3.8 Interaction Invoke V&V Analysis 82

4. Conceptual CACM (*) **84**
4.1 General Metamodel 85
4.1.1 Scope and Purpose 85
4.1.2 Conceptual General Metamodel 85
4.1.3 Conceptual Property Metamodel 88
4.2 System Component Metamodel 89
4.2.1 Scope and Purpose 89
4.2.2 Conceptual Model Definition 89
4.3 Assurance Case Metamodel 90
4.3.1 Scope and Purpose 90
4.3.2 Conceptual Model Definition 90
4.4 Evidence Management Metamodels 93
4.4.1 Scope and Purpose 93
4.4.2 Conceptual Traceability Metamodel 94
4.4.3 Conceptual Managed Artefact Metamodel 97
4.4.4 Conceptual Executed Process Metamodel 103

4.5 Compliance Management Metamodel 105
4.5.1 Scope and Purpose 105
4.5.2 Conceptual Assurance Project Definition 107
4.5.3 Conceptual Process Definition Metamodel 110
4.5.4 Conceptual Standard Definition Metamodel 112
4.5.5 Conceptual Vocabulary Metamodel 118
4.5.6 Conceptual Mapping Definition Metamodel 120

5. Implementation CACM (*) 124
5.1 General Metamodel 124
5.1.1 Scope and Purpose 124
5.1.2 Implementation General Metamodel 124
5.1.3 Implementation Property Metamodel 124
5.2 System Component Metamodel 124
5.2.1 Scope and Purpose 124
5.2.2 Implementation Model Definition 124
5.3 Assurance Case Metamodel 134
5.3.1 Scope and Purpose 134
5.3.2 Implementation Model Definition 134
5.4 Evidence Management Metamodels 147
5.4.1 Scope and Purpose 147
5.4.2 Implementation Traceability Metamodel (AssuranceAsset) 147
5.4.3 Implementation Managed Artefact Metamodel 149
5.4.4 Implementation Executed Process Metamodel 153
5.5 Compliance Management Metamodel 158
5.5.1 Scope and Purpose 158
5.5.2 Implementation Assurance Project Definition 158
5.5.3 Implementation Process Definition Metamodel 160
5.5.4 Implementation Standard Definition Metamodel 160
5.5.5 Implementation Baseline Definition Metamodel 171
5.5.6 Implementation Vocabulary Metamodel 176
5.5.7 Implementation Mapping Definition Metamodel 178

6. Conclusions (*) 179

Abbreviations 181

References (*) 183

Appendix A: Changes since the Predecessor Version D2.3 (*) 185

List of Figures

Figure 1. AMASS Reference (High-Level) Architecture (Prototype P2) 15
Figure 2. Functional Decomposition of the AMASS Platform 19
Figure 3. Architectural Viewpoints for the description of the ARTA 22
Figure 5. Actors of the AMASS Tool Platform 24
Figure 6. Use Cases for “Assurance Project Lifecycle Management” 24
Figure 7. Use Cases for "Assurance Traceability" module 26
Figure 8. Use Cases for “Platform Management” module 28
Figure 9. Use Cases for "System Component Specification", “Architectural Patterns” and “System Architecture Modelling for Assurance” 30
Figure 10. Use Cases for "Contract-based assurance composition" module 35
Figure 11. Use cases for V&V Activities module 40
Figure 12. Use Cases for "Assurance Case Specification" module 46
Figure 13. Use Cases for "Dependability Modelling" module 50
Figure 14. Use Cases for "Contract-based multi-concern assurance" module 51
Figure 15. Use Cases for "System Dependability Co-Analysis/Assessment" module 53
Figure 16. Use Cases for "Evidence Management" module 57
Figure 17. Use Cases for “Tool Integration” module 59
Figure 18. Use Cases for “Compliance Management” module 62
Figure 19. Use Cases for "Product/Process/Assurance Case Line Specification" module 64
Figure 20. Use Cases for "Reuse Assistance" module 66
Figure 21. AMASS Structural View 69
Figure 22. Tool components for Assurance Project Lifecycle Management 69
Figure 23. Tool components for Assurance Traceability 70
Figure 24. Tool components for Platform Management 71
Figure 25. Tool components for System Component Specification 72
Figure 26. Tool components for Assurance Case Specification 73
Figure 27. Tool components for Compliance Management 74
Figure 28. Tool components for Evidence Management 74
Figure 29. Tool components for Tool Integration 75
Figure 30. Tool components for Contracts Management 76
Figure 31. Tool components for Assurance Analysis/Assessment 76
Figure 32. Tool components for Cross/Intra domain reuse 77
Figure 33. Interactions between tool components in the assurance case development scenario 78
Figure 34. Interactions between tool components in the development modular argumentation/architecture scenario 79
Figure 35. Interactions between tool components in the development process based argumentation scenario 79
Figure 36. Interactions between tool components in the Multi-concern co-analysis/assessment scenario 80
Figure 37. Interactions between tool components in the development Cross/Intra Domain Reuse of Base Model scenario 81
Figure 38. Interactions for user access and concurrent evidence management 81
Figure 39. Interactions for evidence information exchange and traceability 82
Figure 40. Interactions for invoke V&V Analysis 83
Figure 41. CACM metamodel views 84
Figure 42. General Metamodel 86
Figure 43. Property Metamodel 88
Figure 44. Conceptual Assurance Case Metamodel diagram 90

Figure 45. Traceability Metamodel 94
Figure 46. Managed Artifact Metamodel....98
Figure 47. Executed Process Metamodel....103
Figure 48. Method Content versus Process in the SPEM 2.0 standard, taken from [28] 106
Figure 49. Assurance Project Metamodel....108
Figure 50. Method Content from the UMA metamodel....111
Figure 51. Process from the UMA metamodel 112
Figure 52. Standard Definition metamodel....113
Figure 53. Conceptual Vocabulary Metamodel 118
Figure 54. Mapping Definition metamodel....120
Figure 55. BlockType 125
Figure 56. Composite BlockType....126
Figure 57. Contract....128
Figure 58. Contract refinement....128
Figure 59. System 130
Figure 60. Artefact and assurance-related entities connections 132
Figure 61. Failure Behaviour 134
Figure 62. Assurance Case class diagram 135
Figure 63. Argumentation Class Diagram....136
Figure 64. The Relationships view diagram....137
Figure 65. Assurance Asset Metamodel....147
Figure 66. Artefact Metamodel (Part 1: Core Model Elements)....150
Figure 67. Artefact Metamodel (Part 1: Core Model Elements....150
Figure 68. Process Metamodel (Part 1: Core Model Elements) 154
Figure 69. Process Metamodel (Part 2: Inheritance Relationships) 154
Figure 70. Assurance Project Metamodel....158
Figure 71. Reference Assurance Framework Metamodel (Part 1: Core Model Elements) 161
Figure 72. Reference Assurance Framework Metamodel (Part 2: Inheritance Relationships) 162
Figure 73. Baseline Definition metamodel (Part 1: Core Model Elements) 172
Figure 74. Baseline Definition metamodel (Part 2: Inheritance Relationship) 173
Figure 75. Vocabulary Metamodel....177

List of Tables

Table 1. Use case “Create Assurance Project” 25
Table 2. Use case “Define Dependability Assurance Project Baseline” 25
Table 3. Use case “Navigate Assurance project repository” 25
Table 4. Use case “Specify Traceability between Assurance Assets” 26
Table 5. Use case “Conduct Impact Analysis of Assurance Asset Change” 27
Table 6. Use case “Configure Access to Assurance Assets” 28
Table 7. Use case “Log in the platform” 28
Table 8. Use case “Concurrent Assurance Information Edition” 29
Table 9. Use case “Specify system architecture at different levels” 31
Table 10. Use case “Monitor status of system specification” 31
Table 11. Use case "Edit an architectural pattern" 33
Table 12. Use case "Instantiate an architectural pattern" 34
Table 13. Use case "Edit parameterized architecture" 34
Table 14. Use case “Provide a configuration constraint for a parameterized architecture” 34
Table 15. Use case “Assign contracts to component” 35
Table 16. Use case “Refine component contracts” 36
Table 17. Use case “Structure properties into contracts” 36
Table 18. Use case “Trace contract to evidence and assurance case” 37
Table 19. Use case “Browse components and associated contracts” 37
Table 20. Use case “Browse component contracts status” 38
Table 21. Use case “Browse Contracts refinement status” 38
Table 22. Use case “Requirements formalisation” 38
Table 23. Use case “Analysis of requirements’ semantics based on their formalisation into temporal logics” 39
Table 24. Use case “Requirements Semantic Analysis” 39
Table 25. Use case “Validate components composition through contracts-based design” 40
Table 26. Use case “Verify contract refinement” 41
Table 27. Use case “Contract Semantic Analysis” 41
Table 28. Use Case “Perform contract-based validation for behavioural models” 42
Table 29. Use Case “Inspect contracts refinement result” 42
Table 30. Use case “Generate fault trees” 42
Table 31. Use case “Perform contract-based fault trees generation” 43
Table 32. Use case “Validate weak contracts” 43
Table 33. Use case "Compare parameterized architecture" 44
Table 34. Use case "Trace requirement validation" 44
Table 35. Use case "Simulation-based fault injection" 45
Table 36. Use case "Monitoring for Functional Verification" 45
Table 37. Use case "Generate documentation from the system model" 45
Table 38. Use case “Define and navigate an assurance case structure” 47
Table 39. Use case “Develop Claims and Links to Evidence” 47
Table 40. Use case “Apply an argument pattern” 48
Table 41. Use case “Reuse an argument pattern” 48
Table 42. Use case “Semi-automatic generation of product arguments” 48
Table 43. Use case “Automatic generation of process arguments” 49
Table 44. Use case “Monitor status of argumentation” 49
Table 45. Use case “Assign to component specification” 50
Table 46. Use case “Specify impact of claims” 51
Table 47. Use case “Navigate, define and develop argument contracts” 51

Table 48. Use case “Inference dependencies” 52
Table 49. Use case “Validate argument contract” 52
Table 50. Use case “Tag multi-concern to contracts” 53
Table 51. Use case “Define and Perform Safety/Security Analysis” 54
Table 52. Use case “Provide results as evidence” 54
Table 53. Use case “Support arguments with result data” 55
Table 54. Use case “Specify evaluate impact of claims” 55
Table 55. Use case “Characterise Artefact” 57
Table 56. Use case “Link Artefact with External Tool” 57
Table 57. Use case “Specify Artefact Lifecycle” 58
Table 58. Use case “Evaluate Artefact” 58
Table 59. Use case “Specify Process Information for Artefacts” 59
Table 60. Use case “Characterise Toolchain” 60
Table 61. Use case “Specify Tool Connection Information” 60
Table 62. Use case “Capture information from standards” 62
Table 63. Use case “Manage Assurance Project” 62
Table 64. Use case “Monitor Assurance Project Status” 63
Table 65. Use case “Define Equivalence Mappings” 63
Table 66. Use case “Define Compliance Mappings” 63
Table 67. Use case “Manage process variability” 64
Table 68. Use case “Manage product variability” 65
Table 69. Use case “Manage assurance case variability” 65
Table 70. Use case “Assist for Cross-System Assurance Assets Reuse” 66
Table 71. Use case “Assist for Cross-Standards Assurance Assets Reuse” 67
Table 72. Use case “Discover Reuse Opportunities by using Standards Equivalences” 68
Table 73. Use case “Reuse Selected Assurance Assets” 68

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 a European-wide open certification/qualification platform, ecosystem and community spanning the largest CPS vertical markets.

AMASS aims at defining an ***AMASS Reference Tool Architecture (ARTA)***. ARTA is specified as a conceptual entity that embodies a common set of tool interfaces/adaptors, working methods, tool usage methodologies, and protocols that will allow any stakeholder of the assurance and certification process to seamlessly integrate their activities (e.g., product engineering, external/independent assessment, and component/parts supply) into tool chains adapted to the specific needs of the targeted CPS markets.

ARTA has been intended to evolve, reflect, and integrate the detailed design performed in the AMASS technical work packages (WP3-WP6).

In this document (D2.4), we present the third version of ARTA, created upon the first version included in D2.2 [1] and the second version included in D2.3 [2] . The document is organised into three different parts:

- **AMASS Reference Tool Architecture.** This part includes the architecture overview and outlines the main ARTA architectural decisions, including a vision of the overall tool architecture. It is based on the architecture proposed in previous phases and its evolution during the third prototype iteration. Since some functionalities such as architectural patterns were not covered in the previous versions of the document, they will be covered along this deliverable.
- **Architectural Views**. This part organises the architecture specification into a set of representative architectural viewpoints, including logical, structural and interactional views. The architecture is intended to evolve during the project life and beyond.
- **CACM definition**. This part describes the Common Assurance & Certification Metamodel (CACM). The CACM aims to provide a basis for common understanding between the different domains and concerns involved in the AMASS project.

This deliverable represents an update of the AMASS D2.3 deliverable [2] released at m18 (September 2017). The sections modified with respect to D2.3 have been marked with an asterisk (*), and the details about the differences and modifications are provided in Appendix A.

1. Introduction

1.1 Context

This section is aimed at recalling the context of the AMASS project as well as the objectives and expected results that pertain to this document.

Embedded systems have significantly increased in number, technical complexity, and sophistication, moving towards open, interconnected, networked systems (such as "the connected car" and the cloud), integrating the physical and digital world, thus justifying the term “cyber-physical systems” (CPS). This “cyber-physical” dimension is exacerbating the problem of ensuring safety, security, availability, robustness and reliability in the presence of human, environmental and technological risks. Furthermore, the products into which these Cyber-Physical Systems (CPS) are integrated (e.g. aircrafts) need to respect applicable standards for assurance and in some areas, the systems even need certification. The dimension of the certification issue becomes clear if we look at the passenger plane Boeing 787 as a recent example – it is reported in [6] that the certification process lasted 8 years and consumed 200.000 staff hours at the FAA, just for technical work. The staff hours of the manufacturer even exceeded this figure, as more than 1500 regulations had to be fulfilled, with evidence reflected onto 4000+ documents. Although aircrafts are an extremely safety-critical product with many of such regulations, the situation in other areas (railway, automotive, medical devices, etc.) is similar.

Unlike practices in electrical and mechanical equipment engineering, CPS do not have a set of standardised and harmonised practices for assurance and certification that ensure safe, secure and reliable operation with typical software and hardware architectures. As a result, the CPS community often finds it difficult to apply existing certification guidance. Ultimately, the pace of assurance and certification will be determined by the ability of both industry and certification/assessment authorities to overcome technical, regulatory, and operational challenges. A key regulatory-related challenge must be faced when trying to reuse CPS products from one application domain in another because they are constrained by different standards and the full assurance and certification process must be applied as if it was a totally new product, thus reducing the return on investment of such reuse decisions. Similarly, reuse is of vital importance in the same domain as well, when trying to reuse CPS products from one project to another.

To deal with the multi-concern nature of present-day critical systems, the complexity due to the proliferation of assessment models and standards, and the presence of hardly interoperable tools, AMASS aims to define and implement a platform that supports those activities required for CPS assurance and certification. AMASS is based on achievements in previous research projects such as SafeCer [4] and OPENCOSS [5]. AMASS plans to integrate and enhance previous results and include new functionalities related to architecture-driven assurance (WP3), multi-concern assurance (WP4), seamless interoperability (WP5), and cross-domain and intra-domain reuse (WP6).

AMASS seeks to support openness of CPS’ technological solutions as a sustainable toolset architecture. To this purpose, the AMASS Reference Tool Architecture (ARTA), documented in this deliverable, establishes and enforces cross-domain and cross-project agreements on toolset architectures, methodological support, interface standards, and interoperability techniques. The task T2.2 *AMASS Reference Tool Architecture and Integration* develops ARTA as an entity that embodies a common set of tool functionalities, user interfaces and tool adaptors. This openness will allow any stakeholder of the assurance and certification process to perform a seamless integration of their activities (e.g., product engineering, external/independent assessment, and component/parts supply) into tool chains adapted to the specific needs of the targeted CPS markets.

1.2 Purpose (*)

This document describes the AMASS Reference tool Architecture (ARTA). This document contains the set of architectural issues, along with the associated architectural decisions; all at appropriate levels of abstraction to make it easy to understand which architectural decisions have been made. The architecture specification has been done in an iterative way as described in the deliverables D2.2 [1] and D2.3 [2] . This document represents the outputs of the third iteration of the specification.

The “Common Assurance & Certification Metamodel (CACM) specification” should be read together with this document. The CACM specification contains the definition of the different concepts that the different ARTA models will use to cover different needs. The CACM specification was included in the D2.2 deliverable “AMASS Reference Architecture (a)” [1]. In the D2.4 deliverable, the current version of the CACM specification is included in Sections 4 and 5. The final version of the CACM specification will be provided in the deliverable D2.9 “AMASS platform validation”.

This document focuses on:

- The ARTA characteristics
- The ARTA boundaries
- The assumptions and restrictions that constrain the design and implementation
- The high-level design of the system in term of its main components and how they should interact with each other.

This document is a guide for further detailed design activities and implementation in the AMASS project that can be used by the AMASS open source community, to be created and coordinated in WP7, task T7.3 Building and Coordination of AMASS Open-Source Community.

This document is also intended as an instrument to ease collaboration among team members in developing the architecture and to help team members easily retrieve the motivation behind technical work-package-specific architectural decisions so that those decisions can be robustly adopted. For example, an architect may make a security-related design decision e.g., by constraining the (type of) data stored in the ARTA and how it can be implemented in the AMASS Platform. This decision may appear to be a burden, but the justification can explain that users do not want to place data outside their toolset and so only want to use the system as a local stand-alone tool or plug-in. The rest of the design must adapt to this restriction.

This document should also inform the AMASS team members about how the system is partitioned or organized so that the team can adapt to the system needs. It also gives a high-level outline of the system and its technical motivations to whoever must maintain and evolve the architecture later (the AMASS community).

This document has three different parts:

1. **AMASS Reference Tool Architecture.** This part includes the architecture overview and describes the main, high-level ARTA architectural decisions including a vision of the overall AMASS Platform. It is based on the view proposed for the core definition and the second prototype (Prototype P1), and it is envisioned for the third prototype (Prototype P2).
2. **Architectural Views**. This part organises the architecture specification into a set of representative architectural viewpoints, including a structural view, a layered view, and an interactional view. The architecture is expected to be evolved beyond the project life.
3. **CACM definition**. This part describes the Common Assurance & Certification Metamodel (CACM). The CACM aims to provide a basis for common understanding between the different domains and concerns involved in the AMASS project.

1.3 Technical Context and Objectives

The objectives we pursue within the ARTA specification are to:

- Specify ARTA as a conceptual entity that embodies a common set of tool interfaces/adaptors, working methods, tool usage methodologies, and protocols that will allow any stakeholders of the assurance and certification/qualification activities to seamlessly integrate their activities (e.g., product engineering, external/independent assessment, component/parts supply) into tool chains adapted to the specific needs of the targeted CPS markets.
- Coordinate the conceptual integration of the AMASS basic building blocks, for each of the four AMASS technical work packages (WP3-WP6).

1.4 Relation to other AMASS Tasks (*)

The ARTA specification in this document is the third iteration result. It is based on previous specifications proposed in the deliverable D2.2 [1] and D2.3 [2]. For this iteration, the inputs have been produced by tasks Tx.2 and Tx.3 (where 'x' is 3 to 6) where they have consolidated the basic building blocks described in the previous iterations and completely addressed the designs of the AMASS functionalities, which aim to fulfil the scientific technological objectives of the AMASS project:

- System Architecture-driven Assurance
- Multi-concern Assurance
- Seamless Interoperability
- Cross/intra-domain Reuse

Special attention has been paid to the definition of an iterative integration plan where the ARTA building blocks are added to the system incrementally. The work was based on the previous specification (D2.2 and D2.3), focusing on this iteration in the communication between the technical areas and addressing completely the functions to fulfil the technical objectives (AMASS-specific building blocks). Within task T2.2, we are following a prototyping approach to design the AMASS basic building blocks and the AMASS-specific building blocks, specifying this way a global integrated platform.

In order to provide a synchronized environment between the different tasks, the two teams created in the AMASS project in the previous iteration have continued working: the *Concept Team* and the *Implementation Team*. These teams serve as interfaces between the technical tasks and the task T2.2. The objective is to provide a quick and smooth connection between the tasks so that the platform design takes into consideration the consistency between the different building blocks.

The aim of the *Concept Team* is to design the CACM as well as to provide guidelines to technical WPs to follow the same strategy and templates for defining the metamodels and functionalities to share among the different areas. This team is also in charge of defining the different metamodels that are part of the CACM and shared by the different work packages, as well as of integrating these metamodels into a coherent and consistent way so that the concepts used in the different technical work packages are captured and shared. The Concept Team is composed by the different conceptual tasks leaders from the technical work packages and T2.2 partners.

The objective of the *Implementation Team* is to discuss cross-WP technical solutions such as database technology options, technologies for the development environment, software licensing, and implementation conventions. The contributors to the Implementation Team are the implementation tasks leaders, the validation task leader, and other implementation-oriented partners. This team works in close cooperation with the Concept Team.

2. AMASS Reference Tool Architecture (ARTA)

2.1 Goals and Philosophy

ARTA proposes a specification for a collaborative tool environment, which aims to support CPS assurance and certification activities. ARTA builds upon OPENCOSS [7], [8], SafeCer [9] and CHESS [23] conceptual, modelling and methodological frameworks and elaborates on new functionalities. More specifically, ARTA not only connects both project achievements but also extends them for architecture-driven and multi-concern assurance, as well as for further cross-domain and intra-domain reuse capabilities and seamless interoperability mechanisms.

ARTA specification was envisioned as a set of blocks that work together to provide all the functionalities.

- AMASS basic building blocks:
  - System Component Modelling Framework
  - Assurance Case Modelling Framework
  - Evidence Management Framework
  - Compliance Management Framework
- AMASS specific tool set:
  - System Architecture-driven Assurance tools
  - Multi-concern Assurance tools
  - Seamless Interoperability tools
  - Cross/intra domain Reuse tools
- Knowledge base conforming to CACM
- Eclipse open-source tools proposed as tool infrastructure
- Open standards for interoperability with external tools.

The ARTA specification is described as an infrastructure on which the AMASS building blocks run and operate in order to perform architecture driven, multi-concern assurance, and cross-domain/concern assurance in an interoperable environment.

The ARTA specification puts high emphasis on modularity and distribution properties in the whole ARTA design, so that it supports connection from different roles and locations to the shared information. In addition, the design must pay special attention to maintainability and flexibility, as the design and implementation shall follow an incremental approach through different iterations.

2.2 Tool Architecture Outline (*)

The ARTA specification aims at being reference in the area of CPS assurance and certification tooling. It is an open architecture with no constraints on the implementation and a solution which provides a customisable assurance assets management infrastructure to support assurance activities along the CPS development lifecycle. Figure 1 provides a high-level overview of ARTA.

The **AMASS Platform Basic Building Blocks** were the result of the first research & development iteration. These building blocks include tools for specification of system components, specification of assurance cases such as structured argumentation trees, evidence management, and compliance management. In addition to these, the basic building blocks include user access management and data management tools, as well as the Common Assurance and Certification Metamodel (CACM). These basic blocks have evolved during the second and the third iteration, which is the last iteration of the AMASS project. All those advanced functionalities build on top of the design solutions detailed in deliverables D3-6.3. The explanation of the implementation of these components is distributed over deliverables D3-6.6. By the

time of the submission of this deliverable, D3.3 and D4.3 have been finalised, while D5.3 and D6.3 continue their conceptual work. The same applies to the last iteration for the implementation deliverables which are ongoing and will be released in the upcoming months. For further details regarding the technical details of that evolution, the reader is referred to Dx.2 [10] [11] [12] [13] and Dx.3 [14] [15] [16] [17], where x equals 3 to 6.

The Dx.2 and Dx.3 deliverables stand for the main Design of the AMASS tools and methods for architecture-driven assurance, multi-concern assurance, seamless interoperability and intra/cross-domain reuse, targeting P1 and P2, respectively.

Supported on the basic building blocks and the second iteration blocks, during this third iteration AMASS has worked on four (4) pillars, which correspond to the specific project Scientific and Technical Objectives (STOs). Their purpose is summarized as follows:

- ***Architecture-Driven Assurance*** **(STO1):** It allows for explicit integration of assurance and certification activities with the CPS development activities, including specification and design. It shall provide support for system components composition in accordance with the domain best practices, guaranteeing that emerging behaviour does not interfere with the whole system assurance.
- ***Multi-concern Assurance*** **(STO2)**: Tool-supported methodology for the development of assurance cases, co-assessment, and contract-based assurance, which addresses multiple system characteristics (mainly safety and security, but also other dependability aspects such as availability, robustness and reliability).
- ***Seamless Interoperability*** **(STO3)**: Open and generically applicable approach to ensure the interoperability between the tools used in the modelling, analysis, and development of CPS, among other possible engineering activities; in particular, interoperability from an assurance and certification-specific perspective, and collaborative work among the stakeholders of the assurance and certification of CPS.
- ***Cross/Intra-Domain Reuse*** **(STO4)**: Consistent assistance for intra- and-cross-domain reuse, or cross-concern, based on a conceptual framework to specify and manage assurance and certification assets.

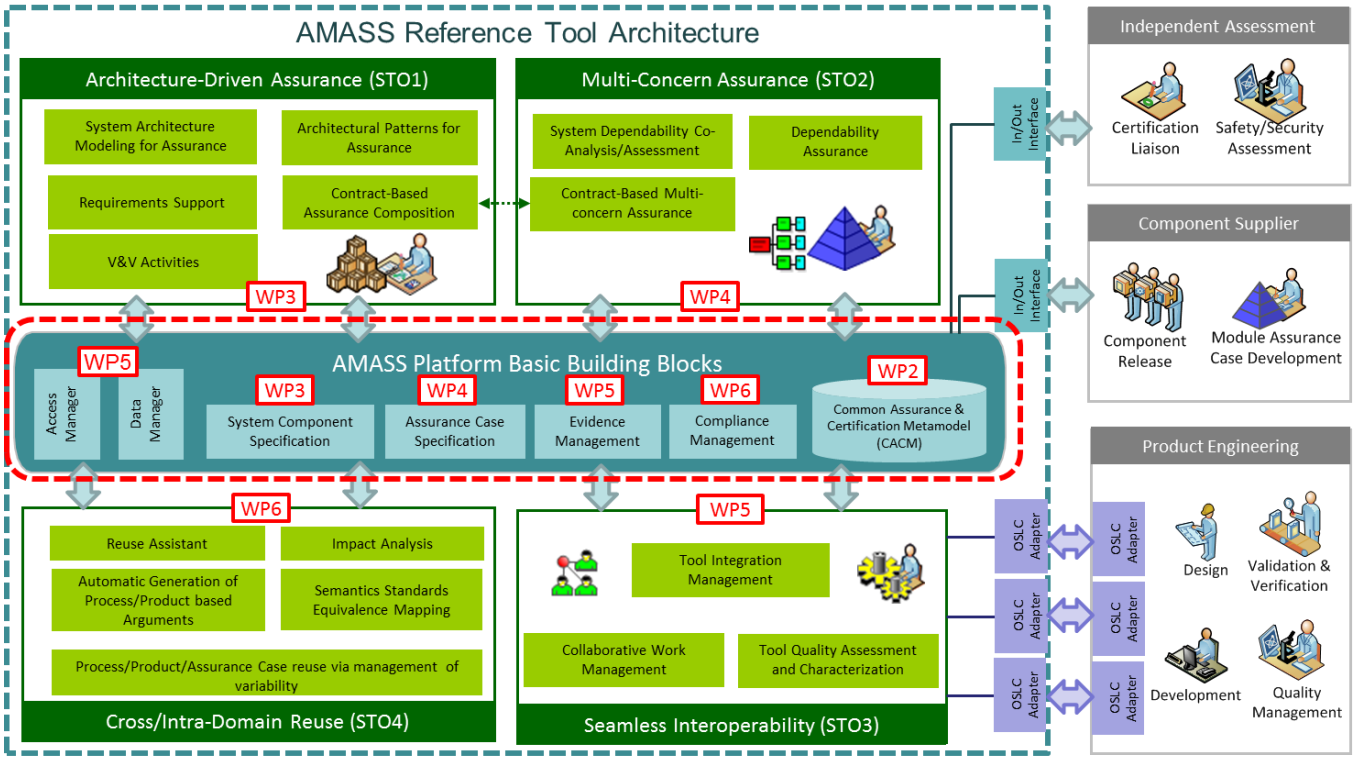


Figure 1. AMASS Reference (High-Level) Architecture (Prototype P2)

In the third iteration the main evolutions have been to (1) consolidate the functionalities of the basic building blocks and (2) address all the advanced AMASS functionalities which are related to the four Scientific Technical Objectives. Further details on the AMASS functionality are presented in Section 3.1.

Table 1 lists the different AMASS functionalities grouped by STOs (cf. Figure 1) for the third and last iteration (P2). Table 1 lists both the AMASS building blocks (blue highlighted) and advanced functionalities (green highlighted).

Table 1. Functionalities of the AMASS Reference Tool Architecture

STO	Functionality Group	Description
Architecture Driven Assurance	**System Component Specification**	This group provides features to allow the modelling of the system architecture specification, in particular, to allow the definition of components as reusable entities, and then the assembly of the components themselves, at any level of the hierarchical architecture, to build/decompose the system.
	System Architecture Modelling for Assurance	This block contains the functionalities that are focused on the modelling of the system architecture to support the system assurance, which are: • Supporting the modelling of additional aspects (not already included in the system component specification) related to the system architecture that are needed for system assurance. • Tracing the elements of the system architecture model to the assurance case. • Generating evidence for the assurance case from the system architecture model or from the analysis thereof. • Importing the system architecture model from other tools/languages. • Functional Refinement**.**
	Architectural	Support for architectural patterns management. This approach helps designer

	Patterns for Assurance	and system architect when choosing suitable solutions for commonly recurring design problems while achieving component reuse. This block contains the functionalities that are focused on architectural patterns to support system assurance, which are: • Management of a library of architectural patterns. • Automated application of specific architectural patterns. • Generation of assurance arguments from architectural patterns application The system component specification should be extended in order to support the specification and instantiation of parametrized architectures. Furthermore, having a contract associated to a specific architectural pattern allows deriving some argumentation fragment automatically. The information regarding the implication of using this pattern is collected in a form of assumption/guarantee (i.e. PatternAssumption and PatternGuarantee). Even if the field of design pattern is large, AMASS focuses on applying its usage on safety-critical systems. Hence, the development of fault tolerance design patterns and its usage for different technologies (also known as technological patterns) are some of the addressed AMASS objectives.
	Contract-Based Assurance Composition	This block, which is also known as *Contract-Based Design for Assurance* introduces, the functionalities that support the contract-based design of the system architecture, which provides additional arguments and evidence for system assurance. These functionalities, also include: • Contract specification, i.e., specification of components' assumptions and guarantees. • Contract-based reuse of components, i.e., a component reuse that is supported by checks on the contracts. • Generation of assurance arguments from the contract specification and validation.
	Activities supporting Assurance Case	This block contains the functionalities that are focused on enriching the assurance case with advanced analysis to support the evidence of the assurance case. These functionalities include the following sub-areas
		Requirements support • Requirements formalisation into temporal logics. • Analysis of requirements' semantics based on their formalisation into temporal logics. • Analysis of requirements based on quality metrics. • Safety requirement derivation based on Model-based Safety Analysis and the creation of the safety concept/definition of safety mechanisms: automatic creation of safety concept from safety contracts (*Contract-based Safety Engineering*).
		V&V Activities or functional verification by simulation and formal methods, safety verification, and validation by analysis techniques and fault injection simulations. • Contract-based verification, i.e. exploiting contracts to verify the architectural decomposition, to perform compositional analysis, and to analyse the safety and reliability of the system architecture. • Automated formal verification (model checking) of requirements on the system design.

		• Model-based specification of fault-injection and analysis of faulty scenarios with simulation (e.g. Sabotage) or model checking (e.g. xSAP) for safety V&V. • Other techniques (e.g. e.g. Component Fault Trees from SysML models) for Model-based safety analysis (e.g. Medini Analyze). • Document generation.
Multi-concern Assurance	**Assurance Case Specification**	This group manages argumentation information in a modular fashion. It also includes mechanisms to support compositional assurance and assurance patterns management.
	Dependability Assurance	This group contains the functionality for creating and structuring the multi-concern assurance case argumentation in an understandable and maintainable way. This includes argumentations targeting various dependability attributes with support of argumentation patterns.
	System Dependability Co-Analysis/Co-Assessment	This group provides functionalities for analysing different quality attributes while taking care of the inter-dependences between them. This is ideally realized by inherently combined Co-Analysis and Co-Assessment methods, which take care of the inter-dependencies within the method. On the other hand, multi-concern assurance can be implemented combining separate processes with mono-concern assurance methods by a workflow tool with a subsequent interaction point activity for treating the mutual dependencies between the quality attributes.
	Contract-based Multi-concern assurance	This group comprises functionalities which contribute to assurance for multiple concerns by two kinds of contracts: on the one hand, component contracts, which target more than one quality attribute. On the other hand, argument contracts, which provide a means for realizing a link between related assurance cases.
Seamless Interoperability	**Evidence management**	This module manages the full lifecycle of evidence artefacts and evidence chains. This includes evidence traceability management and impact analysis.
	Tool Integration Management	This module enables the exchange of data between engineering/assurance tools, e.g. between the AMASS Tool Platform and other tools developed by the AMASS partners.
	Collaborative Work Management	This module allows different users to work at the same time with the same pieces of data, supporting the interaction of the different users.
	Tool Quality Assessment and Characterisation	This module supports the specification and management of tool quality needs for CPS assurance and certification. It is currently supported by the Compliance Management functionality for Cross/Intra-Domain Reuse.
Cross/Intra-Domain Reuse	**Compliance Management**	Functionality related to the management (edition, search, transfer, etc.) of process and standards' information as well as of any other information derived from them, such as interpretations about intents and mapping between processes and standards. This functional group maintains a knowledge database about "standards & processes", which can be consulted by other AMASS functionalities.
	Reuse Assistant	The reuse assistance functionality concerns intra- and cross-domain reuse of assurance and certification assets. This module supports users to understand whether reuse of the assurance assets is reasonable or to determine what further assurance activities (engineering, V&V, or compliance activities) are required to justify compliance in the new scenario.
	Semantics Standards	For analysis of semantics-based equivalence between standards, AMASS extends the OPENCOSS Common Certification Language (CCL) approach by

	Equivalence Mapping	leveraging the SafeCer ontology-based method for representation of standards. Automated means also allows performing an informed gap analysis of the standards and thus mitigates the risk of inappropriate reuse, when a given assurance asset might not appropriately match the requirements of the reuse context.
	Impact Analysis	When an assurance asset is changed, the AMASS platform shall indicate how the change impacts other related assurance assets.
	Process-related reuse via management of variability at process level	Functionality related to the management of variability at process level. This functionality takes as input a process, which needs to be reconfigured, and the new selections desired by the user. As outcome, this functionality generates a new valid re-configuration of the process.
	Product-related reuse via management of variability at product level	Functionality related to the management of variability at product level. This functionality takes as input a product (more specifically, an architectural specification given in CHESSML), which needs to be tailored/reconfigured, and the new selections desired by the user. As outcome, this functionality generates a new valid re-configuration of the architectural specification.
	Assurance Case-related reuse via management of variability at assurance level	Functionality related to the management of variability at assurance case level. This functionality takes as input an assurance case (more specifically, an assurance case developed in OpenCert), which needs to be tailored/reconfigured, and the new selections desired by the user. As outcome, this functionality generates a new valid re-configuration of the assurance case.
	Automatic generation of process-based arguments	This functionality is related to the generation of structured arguments from process models. It supports the strengthening of the safety case via arguments that are aimed at explaining why a process is compliant.
	Automatic generation of product-based arguments	This functionality is related to the generation of structured arguments from contract-based architectural specification. It supports the strengthening of the safety case via arguments aimed at showing why the product is expected to behave safely.

Table 2 lists the main concepts regarding the AMASS infrastructure i.e. Assurance Project Management, Platform Management and Assurance Traceability.

Table 2. AMASS Infrastructure

Functionality Group		Description
Infrastructure	**Assurance Project Lifecycle Management**	This functionality factorizes aspects such as the creation of assurance projects in the ARTA. This module manages a “project repository” which can be accessed by any other ARTA module.
	Platform Management	This is an infrastructure functional module. It includes generic functionality for security, permission and profiles, data storage, visualization, and reporting.
	Assurance Traceability	This is an infrastructure functional module. It includes generic functionality for traceability management and impact analysis.

Figure 2 illustrates the functional decomposition of the AMASS Platform. This functional decomposition is further analysed in Section 3.

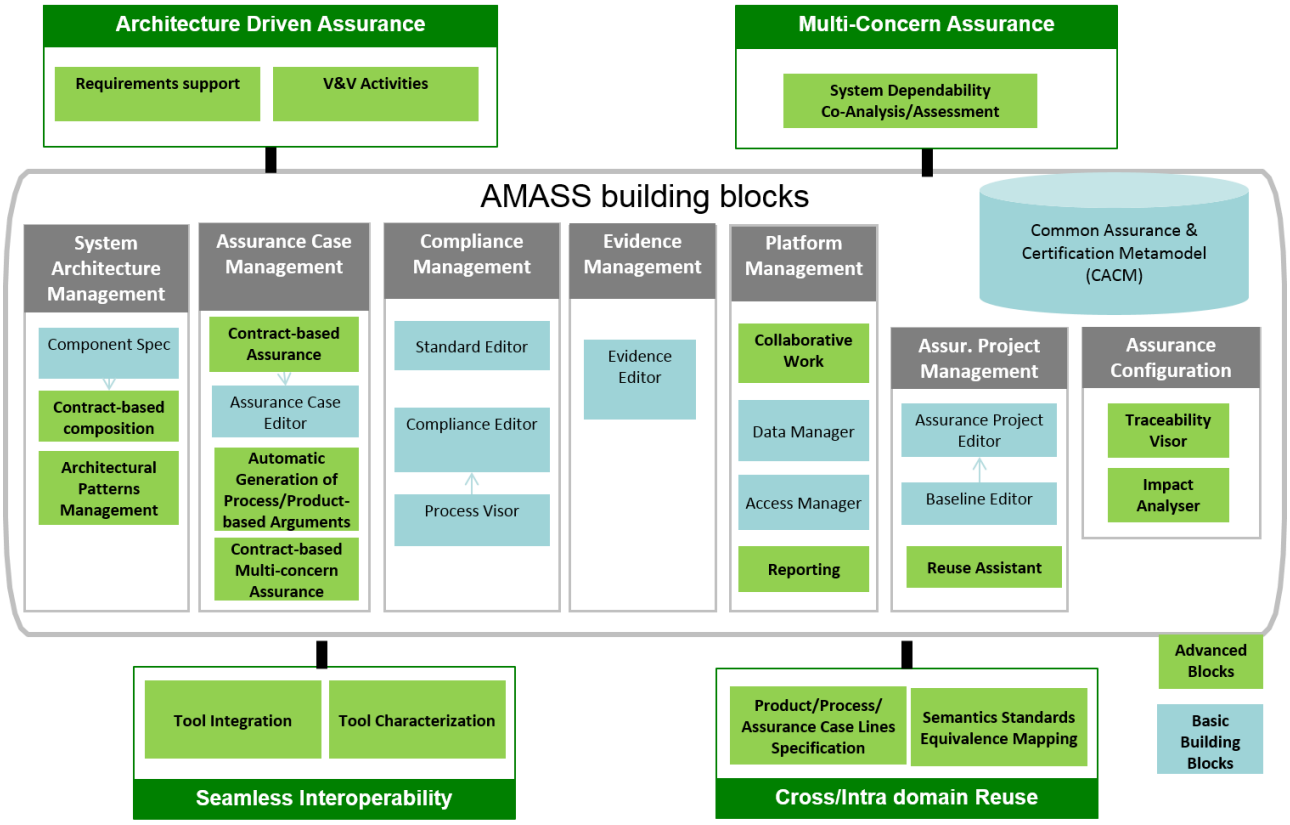


Figure 2. Functional Decomposition of the AMASS Platform

2.3 Stakeholders

In this section, we present a general view of the different stakeholders for the ARTA. We can identify the following main groups:

- Manufacturer
- Supplier
- Assessor and Authorities
- Tool providers

Manufacturer

They are the interface with the user groups and organizations. They are partly responsible to introduce the ARTA envisioned tools to the public. This includes the system manufacturers from the different domains, system responsible, engineers, assurance assessors to carry on the assurance activities, evidence collection, and so on.

The **Project Manager** is the role that works on a compliance and assurance-based project where a product (system or component) needs to be assessed as acceptably dependable (safety, security or other dependability properties). The Manager will use an implementation of the ARTA (e.g. the AMASS Tool Platform) to check the status of the project's goals within the planned budget, time, and resources.

The **Assurance Engineer** is the role responsible for executing the different V&V and assurance activities e.g., create and/or collect the evidence to demonstrate that the product is acceptable safe/secure. An assurance engineer can be split into safety and security engineer roles.

The **Assurance Manager** is responsible to show compliance with a particular standard and argue the safety/security of the product in an assurance case, or demonstrate the properties of a component or system that are required for an assurance case. The Assurance Manager will use the ARTA platform to plan, structure, view, review and assess the system structure and arguments or modules, sometimes by composing pre-existing arguments, and reusing arguments and evidence relating to reusable components. In the same way that an assurance engineer can be split in two groups (i.e. safety and security engineer), an assurance manager can be subdivided into safety and security manager.

The **Internal Assessor** is responsible for assessing the adequacy of the evidence and assurance 'package', in terms of demonstrating the safety/security of the system under consideration.

Supplier

Suppliers are system and component development organisations that are the functional and financial beneficiaries of the AMASS tools. A supplier can have the same kind of actors as described for the manufacturer stakeholder (project manager, assurance manager, assurance engineer and internal assessor). They can either benefit from the use of the AMASS, by implementing tools or integrating some of the available tools within their own products or by using the tools and provide composable assurance assets (e.g. assurance cases, composable evidence, component modules design in the architecture, etc.).

Assessor and Authorities

The **Authority** may be of three types:

- The National Safety/Security Authority is a generic placeholder for the various national bodies responsible for safety/security in a particular domain. These bodies are answerable to the national governments. They are included here as political beneficiaries from the AMASS tools, since improved visibility of assurance is a benefit.
- The European Safety/Security Authority is a generic placeholder for the European overseers of overall transport safety in the aerospace and railway domains. As with the National Safety/Security Authorities, they will benefit politically from the enhanced visibility of assurance.
- The Regulator is a generic placeholder for the safety/security regulatory and certification bodies in the aerospace and railway domains (automotive manufacturers do not answer directly to a certification body).

The **Assurance Assessor** is responsible for assessing the adequacy of the evidence and assurance 'package' provided by the manufacturers, in terms of justifying the safety/security of the system or component under consideration. Depending on the domain, and on the nature of the system under consideration, the assurance assessor may be independent of the manufacturing organisation. The safety/security assessor will use the AMASS tools to view workflows, arguments, compliance checklists and evidence artefacts related to the system or component.

Tool providers

The **tool provider** creates tools for manufacturing CPS or supporting assessment of these systems. Their objective is to create competitive tools that support the manufacturers and assessors in their goals. They will either implement possible alternatives for the ARTA or connect with the AMASS tools as it clearly provides an added value for manufacturers and assessors in achieving their goals.

Please note that the group of AMASS stakeholders is refined in the "logical view" in Section 3.1.

2.4 AMASS Tool Platform (*)

The **AMASS Tool Platform** (or the AMASS Platform) is a specific implementation of ARTA, with capability for evolution and adaptation, which will be released as an open technological solution by AMASS through

the OpenCert project [18]. The AMASS Tool Platform is a collaborative tool environment that supports CPS assurance and certification.

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

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

AMASS has planned three prototype iterations and the results are being released in the OpenCert project hosted by the PolarSys working group [19]:

1. During the **first prototyping** iteration (Prototype Core), the AMASS Platform Basic Building Blocks were aligned, merged and consolidated at TRL4 (Technology Readiness Level - 4).
2. During the **second prototyping** iteration (Prototype P1), the previous Basic Building Blocks were improved and the first version of the AMASS-specific Building Blocks were developed and benchmarked at TRL4.
3. Finally, at the **third prototyping** iteration (Prototype P2), all AMASS building blocks will be integrated in a comprehensive toolset operating at TRL5.

Each of these iterations has the following three prototyping dimensions:

- ***Conceptual/research development***: development of solutions from a conceptual perspective.
- ***Tool development***: development of tools implementing conceptual solutions.
- ***Case study development***: development of industrial case studies using the conceptual and tooling solutions.

This deliverable reports the **ARTA** design of the third prototype iteration (Prototype P2), which covers the basic building blocks, and a second version of the AMASS-specific building blocks (which provide a solution to the AMASS scientific technological objectives: Architecture-driven Assurance, Multi-concern Assurance, Seamless Interoperability and Cross/Intra Domain Reuse).

3. Architectural Views

The architectural description of the ARTA focuses on the concepts of *view* and *viewpoint* [20]. A view is "a representation of a whole system from the perspective of a related set of concerns based on a viewpoint". A viewpoint is a "specification of the conventions for constructing and using a view", e.g. technology viewpoint, information model viewpoint. A set of architectural viewpoints that describe the ARTA from a high-level design perspective have been selected. In this chapter, we include the different architectural views that any implementation of the ARTA should fulfil. For validation purposes, the AMASS platform will be used as the implemented version of the ARTA.

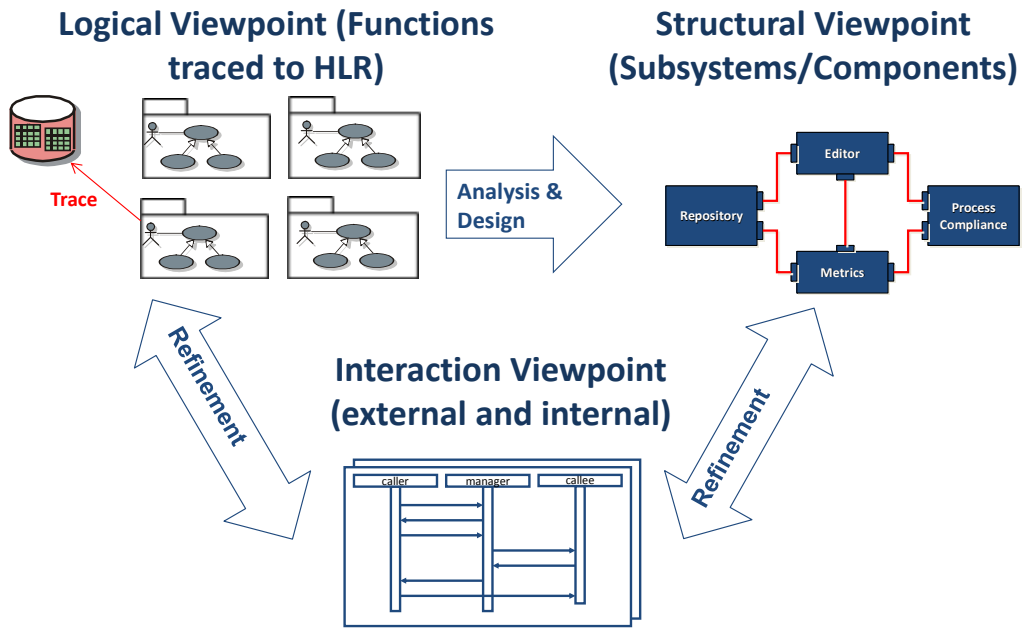


Figure 3. Architectural Viewpoints for the description of the ARTA

The ARTA architecture specification has three viewpoints (see Figure 3):

- **Logical Viewpoint (see Section 3.1)**. The ARTA Logical Viewpoint shows the functionality that will be required to fulfil the High Level Requirements (HLR) [21]. In this document, the Logical Viewpoint is shown as UML Use Case diagrams that contain actors and use cases with interaction flows.
- **Structural Viewpoint (see Section 3.2)**. The AMASS Structural Viewpoint defines a set of building blocks (tool modules), their services and their data. In this document, we describe the Structural Viewpoint by using UML package and component diagrams.
- **Interaction Viewpoint (see Section 3.3).** The AMASS Interaction Viewpoint describes the scenarios of interaction between the user and the ARTA platforms and internally between tool modules. We have described the interaction viewpoint by using UML sequence diagrams.

In our approach of architectural description, we use a "layered" pattern to handle functionality at different levels of service reusability and abstraction. The layered pattern structures the system description into groups of modules that form layers on top of each other. In this pattern, upper layers use services of the layers below only (never above).

3.1 Logical View

The Logical View description organizes the ARTA specification into different use case groups. We have derived functional groups from an analysis of Case Studies description [22] and Requirements Specification [21]. The analysis consisted of the following points:

- Selection of requirements [21] related to the “third prototype” (Prototype P2) in the information workflow handled by AMASS. Selection of needs from the case studies [22] updating or reading common set of data. For instance, use cases managing evidence have been grouped in a single functional group.
- Selection of the logical functionalities analysed and conceptual design included in the technical deliverables Dx.2 [10][11][12][13] and Dx.3 [14][15][16][17] (where x=3-6).
- Transversal functionalities have been factorized in common modules. For instance, access management and assurance project lifecycle management have resulted in independent functional groups.

During the third prototype iteration, all the functionalities described in the ARTA have been fully addressed. For instance, comparing with the previous D2.3, use cases related to architectural patterns for assurance, functional verification by monitors and simulation-based fault injection have been defined.

The technical Dx.3 (where x=3-6) deliverables have improved the overall understanding of the general platform.

Table 1 lists the main AMASS functionalities while Table 2 depicts the main concepts addressing the AMASS infrastructure i.e. Assurance Project Management, Platform Management and Assurance Traceability.

To describe each of the proposed functionality groups, use case diagrams are utilised. A use case is a list of steps, typically defining interactions between a role (known as an "actor") and a system (the AMASS platform in our case), to achieve a goal. Figure 4 defines the set of actors which interact with the AMASS platform. The actors are based on stakeholders (Section 2.3) with some refinements:

- **Management**. It stands for Project and Assurance Manager, which is an AMASS-specific actor artificially created to represent a manager who is in charge of managing all the processes and activities involved in the AMASS platform usage. This also includes an IT Manager who is in charge of managing and setting the AMASS tool platform, as an IT infrastructure.
- **Engineers**. Any actor involved in the execution of development, V&V and safety-security analysis activities. We separate safety and security engineers, since some activities may need to distinguish according to the targeted concern (safety and security). The process Engineer is a subtype of a development engineer who is in charge of defining the development processes and ensure the process is being followed.
- **Assessors**. Two kinds of assessors: internal to the company and external or independent assessor. In this iteration they are not distinguished in the proposed logic, however, they could have some implications regarding accessibility to certain confidential assurance assess.

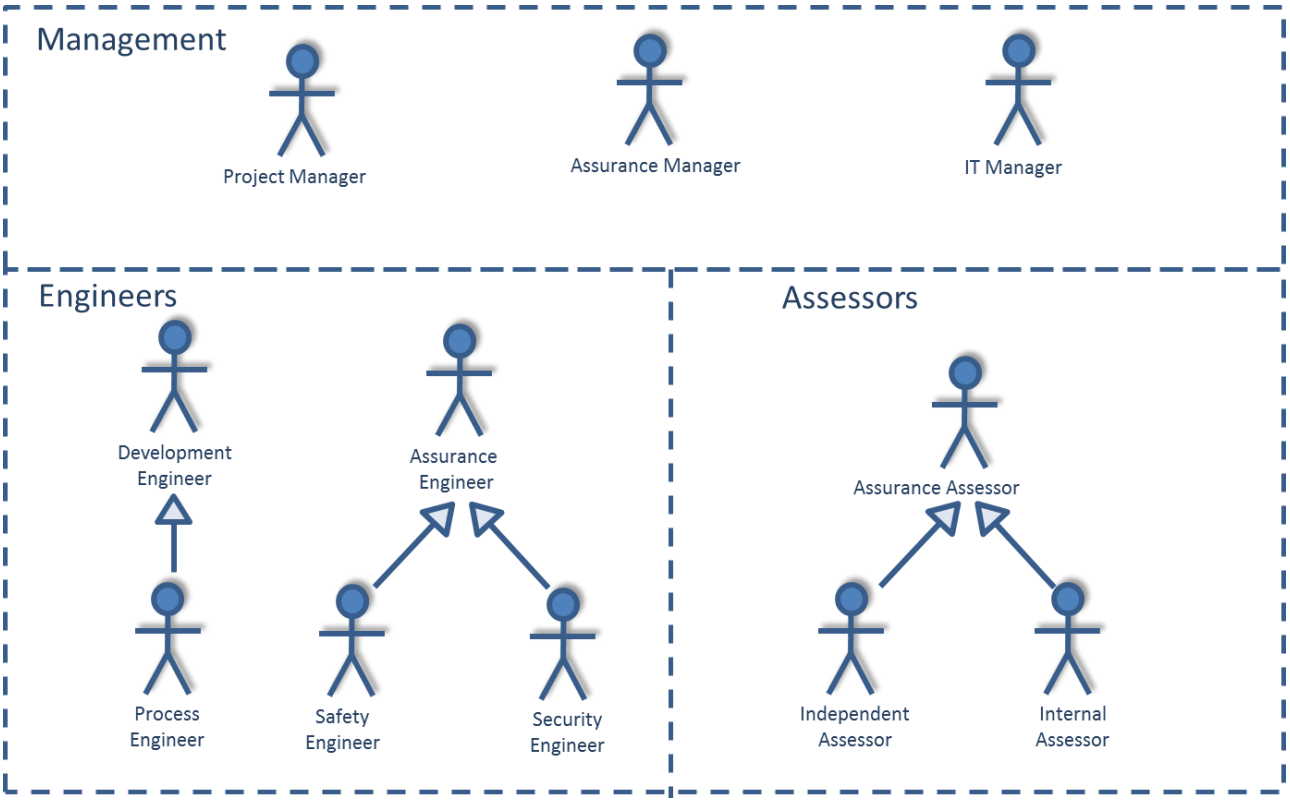


Figure 4. Actors of the AMASS Tool Platform

3.1.1 Infrastructure Use Cases

3.1.1.1 Assurance Project Lifecycle Management Use Cases

This functionality includes aspects such as the creation of assurance projects locally in the AMASS Platform and any project baseline information that may be shared by the different functional modules. This module manages a "project repository", which can be accessed by the other AMASS modules.

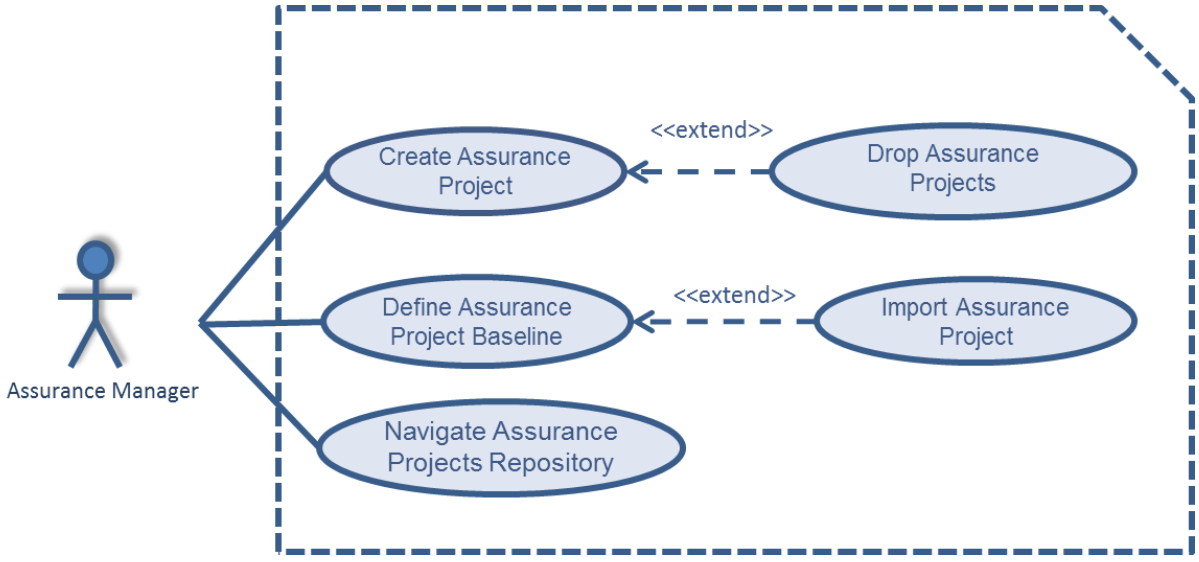


Figure 5. Use Cases for "Assurance Project Lifecycle Management"

Table 1. Use case "Create Assurance Project"

Use Case	Create Assurance project
Functionality Description	The system should allow users to create a kind of "container" for the whole information related to a given safety assurance project.
Actors	Assurance Manager
Assumptions	General Information about the project must be available: general timing, responsible person in the system, client, product and type of product under assurance, etc.
Post-conditions	None
Steps	1. Create a new project defining any dependency with other projects 2. Specify general project information (general timing, responsible person in the system, client, product and type of product under assurance, etc.)
Variations	#Drop Assurance Project: The system could delete information about a whole assurance project from the repository
Non-functional	None
Requirements	None

Table 2. Use case "Define Dependability Assurance Project Baseline"

Use Case	Define Dependability Assurance Project Baseline
Functionality Description	The system should allow users to define a technical information baseline about a given project, including, standards scope, compliance means, and justifications on any project decisions of company process tailoring.
Actors	Assurance Manager
Assumptions	Technical information about the project must be available: project plans, dependability/safety/certification plans, standards scope, and means of compliance (agreed with authorities.
Post-conditions	None
Steps	1. Define project structure (into sub-projects if needed). 2. Define the scope of standards for this project (phases, activities, etc.) and/or sub-projects if needed. 3. Define the compliance means (evidence to be presented for compliance) 4. Specify any justification on compliance means
Variations	#Import an assurance project: the system lets the user import the information for this use case, from an external file created with the process editor.
Non-functional	None
Requirements	None

Table 3. Use case "Navigate Assurance project repository"

Use Case	Navigate Assurance projects repository
Functionality Description	The system should allow users to navigate along the assurance project repository.
Actors	Assurance Manager
Assumptions	The assurance project repository previously exists.
Post-conditions	None
Steps	1. Open the Assurance Project. 2. Navigate through the different assurance project elements.
Variations	None
Non-functional	None

Requirements	

3.1.1.2 Assurance Traceability Use Cases

Traceability can be defined as the existence of a relationship between two artefacts relevant to a system's lifecycle; e.g. between a requirement and the test cases that validate it. The Assurance Traceability Use Cases focus on cross-cutting traceability aspects that other AMASS areas might need, i.e. traceability between different assurance assets: between evidence artefacts, between an assurance case and the evidence artefacts that the assurance case reports, between a component contract and its evidence of satisfaction, etc. Traceability is not limited only to the specification of relationships but also deals with other activities that exploit it, specifically, with change impact analysis.

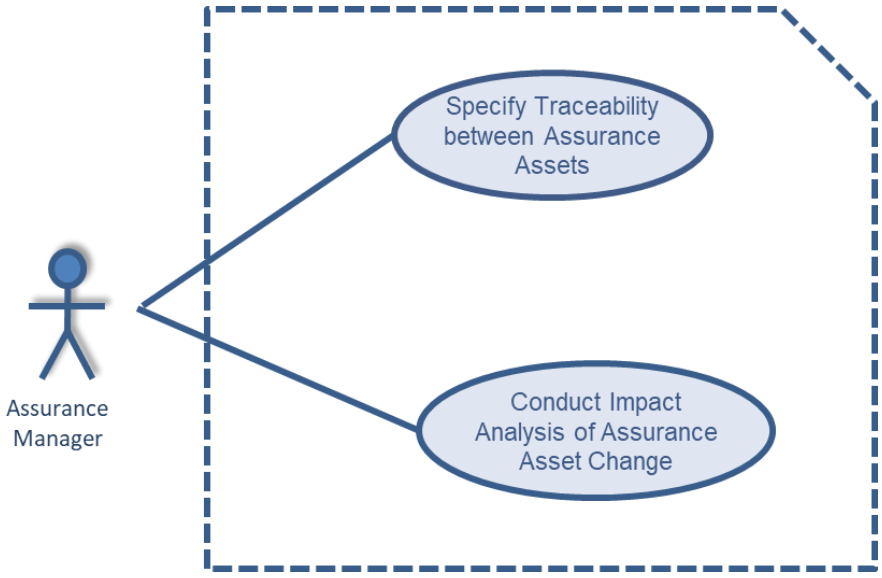


Figure 6. Use Cases for "Assurance Traceability" module

Table 4. Use case "Specify Traceability between Assurance Assets"

Use Case	Specify Traceability between Assurance Assets
Functionality Description	The AMASS Platform shall: (1) allow an Assurance Manager to specify relationships between evidence artefacts; (2) display the chains of evidence to which an evidence artefact belongs; (3) analyse the quality of the relationships between evidence artefacts (e.g. completeness and correctness). When specifying relationships for an evidence artefact, the system shall suggest evidence artefacts to which the first evidence artefact might relate. The same kind of features could be used for traceability of other assurance assets.
Actors	Assurance Manager
Assumptions	Two Assurance Assets have been created.
Post-conditions	The trace chain of the specified traceability is shown.
Steps	1. The Assurance Manager selects an Assurance Asset 2. The Assurance Manager adds a Trace Link to the Asset 3. The Assurance Manager indicates the Assurance Asset or Assets that corresponds to the target of the Trace Link
Variations	# The AMASS Platform indicates Assurance Assets that could correspond to the target of the Trace Link # The Assurance Manager indicates the Change Effect Kinds of the Trace Link (None, To Validate, To Modify, Modification, Revocation; see details in CACM definition in D2.2)

	# The AMASS Platform shows information about the quality of the Trace Links # The AMASS Platform shows possibly missing Trace links # The Assurance Manager evaluates the quality of a Trace Link
Non-functional	None
Requirements	WP5_EM_002, WP5_EM_008, WP5_EM_009, WP5_EM_012

Table 5. Use case "Conduct Impact Analysis of Assurance Asset Change"

Use Case	**Conduct Impact Analysis of Assurance Asset Change**
Functionality Description	When an evidence artefact is changed, the AMASS Platform shall indicate how the change impacts other evidence artefacts. The AMASS Platform shall allow an Assurance Manager to indicate what evidence artefacts are actually impacted by the changes to a given evidence artefact. The same kind of features could be used for change impact analysis of other assurance assets.
Actors	Assurance Manager
Assumptions	Several Assurance Assets have been created. An Assurance Asset has been changed. Traceability between the Assurance Assets has been specified, including change effects in the trace links
Post-conditions	The potential change impact matches the information of the trace links between Assurance Assets.
Steps	1. The AMASS Platform conducts an impact analysis for the change of an Assurance Asset 2. The AMASS Platform shows the potential change impacts 3. The Assurance Manager selects the relevant change impacts 4. The AMASS Platform records the change impact selection and propagates accordingly (i.e. determine impacts resulting from the selection in step 3).
Variations	None
Non-functional	None
Requirements	WP5_EM_003, WP5_EM_011

3.1.1.3 Platform Management Use Cases

This functionality factorises features that provide access to users and secure the data stored in the platform. It ensures the authentication and availability of the data stored in the platform. The functionality also addresses the possibility of concurrent, collaborative work among different users.

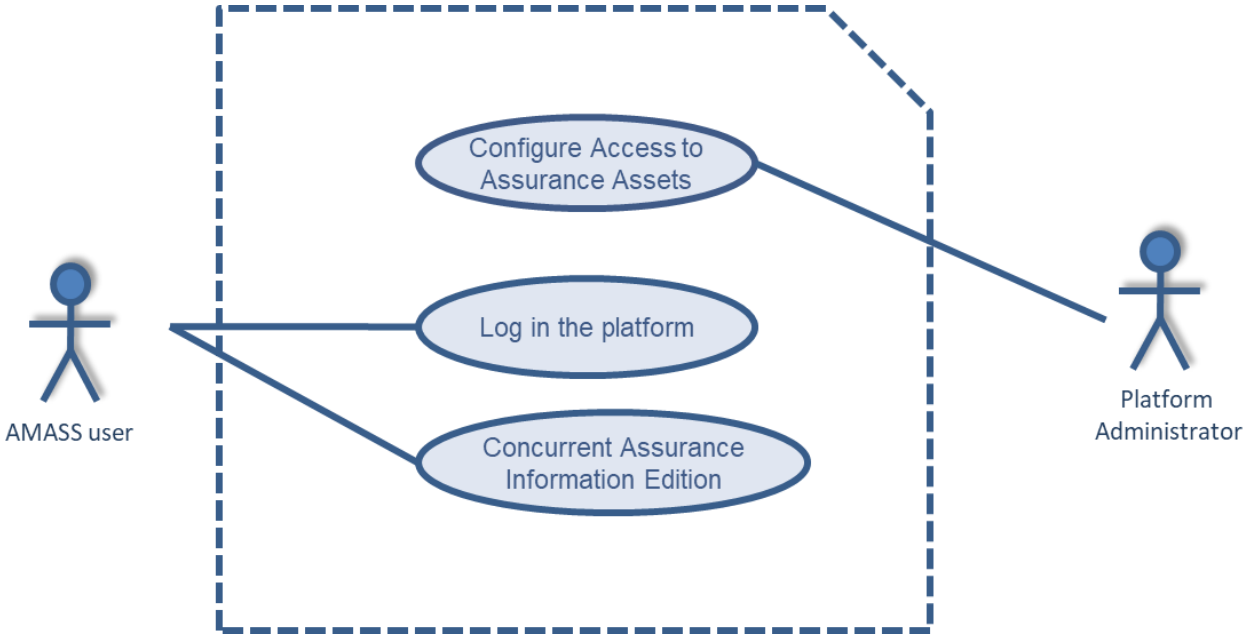


Figure 7. Use Cases for "Platform Management" module

Table 6. Use case "Configure Access to Assurance Assets"

Use Case	Configure Access to Assurance Assets
Description	The AMASS Tool Platform shall provide users with different options for data access and for action permission, to allow users to have different profiles (roles) for Platform access, and to belong to different access rights groups depending the role they are executing at that moment. To these ends, it is necessary that a platform administrator makes the necessary access configurations.
Actors	Platform Administrator
Assumptions	None
Pre-conditions	None
Post-conditions	The AMASS user information is recorded in the Platform.
Steps	1. The Platform administrator creates a new AMASS user account for the AMASS Tool Platform 2. The Platform administrator assigns a user profile group to the AMASS user account 3. The Platform administrator grants additional access permission to the AMASS user account
Variations	# The Platform administrator creates a new user profile group and specifies access permission for the group
Non-functional Requirements	None
	WP5_AM_002, WP5_AM_004, WP5_AM_005

Table 7. Use case "Log in the platform"

Use Case	Log in the platform
Description	The AMASS Tool Platform shall require users to be authenticated for Platform access. This in turn will allow the Platform to maintain a log with all the actions performed by the users. The AMASS Tool Platform shall also provide a secure standard API for data access.
Actors	AMASS user: Project Manager, Assurance Manager, Platform Administrator Development Engineer, Assurance Engineer, Assurance Assessor

Assumptions	The user credentials are already defined in the platform.
Pre-conditions	An account has been created for the AMASS user.
Post-conditions	The AMASS user can access the AMASS Tool Platform data and services according to the access permission granted.
Steps	1. The AMASS user indicates its user credentials 2. The AMASS Platform validates the credentials 3. The AMASS user accesses the AMASS Tool Platform
Variations	# If the user name or password are incorrect, the AMASS user is notified about it
Non-functional	# The AMASS user log-in shall use a secure standard API
Requirements	WP5_AM_001, WP5_AM_003, WP5_DM_008

Table 8. Use case "Concurrent Assurance Information Edition"

Use Case	**Concurrent Assurance Information Edition**
Description	During CPS lifecycle, it is usually necessary that several people work together to perform CPS lifecycle activities. Support to concurrent work in this collaboration can be provided, so that different users can access and edit assurance information at the same time. Examples of collaborations that can be taken into account include: • systems engineers, safety engineers, and security engineers for system analysis • systems engineers, safety engineers, and security engineers for system modelling • systems engineers, assurance managers for management of compliance with standards and of process assurance • assurance managers and assurance engineers for re-certification needs & consequences analysis • systems engineers for system V&V • systems engineers, safety engineers, and security engineers for model-based systems engineering • assurance managers and systems engineers for assurance evidence management • systems engineers and assurance managers for product reuse needs & consequences analysis • assurance managers and assurance engineers for assurance case specification • assurance managers for compliance needs specification • assurance managers, assurance engineers, and assurance assessors for assurance assessment • assurance managers, assurance engineers, and assurance assessors for compliance assessment In addition, the AMASS Tool Platform shall manage metrics and measurements about collaborative work.
Actors	AMASS user (several)
Assumptions	None
Pre-conditions	The AMASS users have logged in
Post-conditions	The AMASS Tool Platform is consistent
Steps	1. An AMASS user accesses some data 2. Another AMASS user accesses the same data 3. Both users are notified of the concurrent access 4. The first AMASS user changes some data 5. The second AMASS user is notified of the data change
Variations	# If a (possible) conflict arises from concurrent data access, the AMASS users will be notified about it. # The AMASS users can access information about the collaborative work performed

Non-functional Requirements	None				
	WP5_DM_001,	WP5_DM_002,	WP5_DM_003,	WP5_DM_004,	WP5_CW_001,
	WP5_CW_002,	WP5_CW_003,	WP5_CW_004,	WP5_CW_005,	WP5_CW_006,
	WP5_CW_007,	WP5_CW_008,	WP5_CW_009,	WP5_CW_010,	WP5_CW_011,
	WP5_CW_012,	WP5_CW_013			

3.1.2 Architecture-driven Assurance Uses Cases (*)

This section addresses the uses cases which are related to STO1 Architecture-Driven Assurance. This stands for *System Component Specification*, *System Architecture Modelling for Assurance*, *Architectural Patterns for Assurance*, *Requirements Support* and *V&V Activities*. Requirements support and V&V activities are part of the so-called "Additional Activities supporting the Assurance Case" in D1.5 [3].

The new use cases related to architecture-driven assurance in terms of System Component Specification, System Architecture Modelling for Assurance and Architectural Patterns for Assurance and are highlighted in yellow in Figure 8 . The rest of the use cases regarding architectural-driven assurance are tackled in Figure 9 and Figure 10.

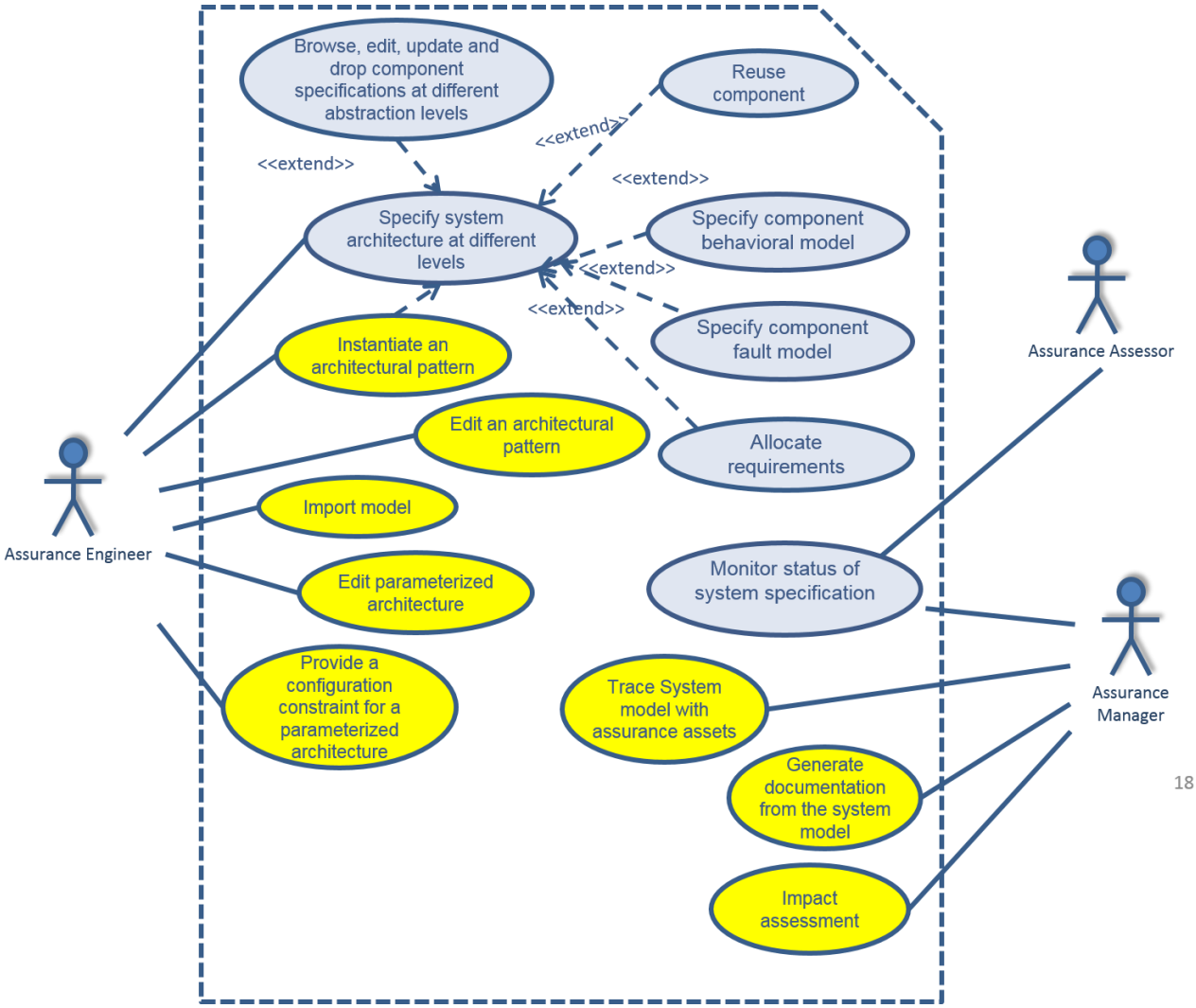


Figure 8. Use Cases for "System Component Specification", "Architectural Patterns" and "System Architecture Modelling for Assurance"

3.1.2.1 System Component Specification Use Cases

The use cases for system component specification deal with the possibility to provide and browse information about the architecture of the system of interest and how the entities of the architecture itself can be related to the assurance case. In particular, regarding the design of the system, a key role is played by the concept of **component**, a reusable entity which can be used, in a top-down or bottom-up process, to specify the architecture of the entire system in a hierarchical way, at different abstraction levels.

Table 9. Use case "Specify system architecture at different levels"

Use Case	Specify system architecture at different levels (e.g., component interfaces, subcomponents, connections)
Functionality Description	System component Specification Editor shall allow the user to create, update, modify, and delete the architectural entities.
Actors	Assurance engineer
Assumptions	The actor has proven knowledge about model-based design.
Post-conditions	The specified architecture is compliant with the CACM.
Steps	1. Create a new system/component project and model. 2. Open the model 3. Use the model-based editor facilities to create, update, modify, delete the entities.
Variations	#Browse along the different abstractions levels 1. Select the component model diagram 2. Use the model editor explorer and diagrams tabs to navigate the contents 3. Edit, modify, update the contents or drop the diagrams. # Specify component behavioural model 1. The actor uses the system model editor functionalities to create, associate to the component, and model the state machine which defines the component behaviour. # Specify component fault model 1. The actor uses the system model editor functionalities to create, associate to the component, and model the fault behaviour for the component itself. # Reuse Component 1. When defining the system architecture, the actor will be able to reuse previous component model, together with its previously specified behavioural models, state machine, fault models, component contracts, associated arguments and evidence. # Allocate requirements 1. The user will specify allocated requirements to the selected component.
Non-functional Requirements	None
	WP3_SC_001, WP3_SC_006, WP3_SC_007, WP3_SC_005, WP3_SC_002

Table 10. Use case "Monitor status of system specification"

Use Case	Monitor status of system specification
Functionality Description	System component Specification Editor shall allow users to view a summary of the system specification highlighting the components with no specification completed.
Actors	Assurance Manager, Assurance Assessor
Assumptions	The system architecture main component shall be already defined, together with associated contracts.
Post-conditions	The report is generated.

Steps	The user asks for the system specification report. The report shall include: • The number of contracts not fulfilled (not validated with arguments and/or not supported by evidence).
Variations	None
Non-functional	None
Requirements	WP3_SC_001, WP3_CAC_012, WP3_VVA_007

3.1.2.2 System Architecture Modelling for Assurance Use Cases

This section addresses the uses cases which are related to System Architecture Modelling for Assurance.

Table 3. Use case "Trace system model with assurance assets"

Use Case	Trace system model with assurance assets
Functionality Description	The system editor shall allow the user to specify the links between the system components and assurance case-related information.
Actors	Assurance Manager
Assumptions	Components and assurance case-related information to be trace shall be already defined.
Post-conditions	The links conforms to the CACM specification.
Steps	1. The user selects the component to trace. 2. The user selects the fragments for the assurance case to trace. 3. The tool allows the user to create the trace for the selected entities.
Variations	None
Non-functional	None
Requirements	WP3_SAM_001

Table 4. Use case "Import model"

Use Case	Import model
Functionality Description	The System Component Specification Editor shall allow the user to import a model from an external system editor.
Actors	Assurance Engineer
Assumptions	The system model to be imported is compatible with the system component metamodel defined within the CACM
Post-conditions	The imported model conforms to the CACM specification.
Steps	1. The user selects the system model (or the package) where the model has to be imported, by using the system editor coming within the ARTA 2. The user selects the “import model” from the menu command associated to the editor 3. The user selects the model to be imported 4. The tool creates the imported entities in the current system model. In case of errors, the system notifies to the user such errors. 5. Optionally, the tool creates the diagrams related to the imported entities.
Variations	None
Non-functional	None
Requirements	WP3_SAM_004

Table 5. Requirements allocation

Use Case	Requirements Allocation
Functionality Description	The system must provide the capability for allocating requirements to parts of the component model. More in general, requirements traceability shall be enabled (Functional Refinement).
Actors	Assurance Engineer
Assumptions	The requirements should be previously defined.
Post-conditions	None
Steps	1. The user opens system component editor. 2. The user selects the component. 3. The user allocates a certain requirement to the component.
Variations	None
Non-functional	None
Requirements	WP3_SC_005

Table 6. Use case "Impact assessment"

Use Case	Impact assessment
Functionality Description	The system component shall provide the capability for a component change impact analysis
Actors	Assurance Manager
Assumptions	The component to be changed has contracts associated and traceability links to the assurance case and evidence
Post-conditions	None
Steps	1. The user opens the traceability view 2. The user selects the component 3. The traceability view shows the contracts associated to the components, so the requirements formalized by the contracts themselves. Moreover, the view shows the assurance case and the evidence traced to the component.
Variations	None
Non-functional	None
Requirements	WP3_SAM_002

3.1.2.3 Architectural Patterns for Assurance Use Cases

This section addresses the uses cases which are related to Architectural Patterns for Assurance.

Table 11. Use case "Edit an architectural pattern"

Use Case	Edit an architectural pattern
Functionality Description	System component Specification Editor shall allow users to create and edit an architectural pattern, to be instantiated later in a given system architecture.
Actors	Assurance Engineer
Assumptions	None
Post-conditions	The architectural pattern is available to be instantiated in a given system model.
Steps	1. Create a new project-model, or open an available model 2. Create a new component diagram 3. Use the model-based editor facilities to create a composite component tagged as pattern 4. Create a dedicate diagram to design the roles of the patterns and the connections between the roles
Variations	None

Non-functional	None
Requirements	WP3_APL_002, WP3_APL_003

Table 12. Use case "Instantiate an architectural pattern"

Use Case	**Instantiate an architectural pattern**
Functionality Description	System component Specification Editor shall allow users to instantiate an architectural pattern previously defined in a given system architecture.
Actors	Assurance Engineer
Assumptions	None.
Post-conditions	The architectural pattern is available in a given model.
Steps	1. Open the system model where the pattern has to be instantiated 2. Import the model where the pattern has been defined, if different from the current system model 3. Ask the tool to instantiate an existing pattern 4. The tool shows the patterns defined in the current and imported models 5. Select the pattern to be instantiated 6. Use the model-based editor facilities to bind the roles and connections defined for the pattern to the entities available in the current system 7. The tool creates an entity in the system model representing the instantiated patterns and store the information about the bindings.
Variations	None
Non-functional	None
Requirements	WP3_APL_001

Table 13. Use case "Edit parameterized architecture"

Use Case	**Edit parameterized architecture**
Functionality Description	The System Component Specification Editor shall allow the user to specify the parameterized architecture
Actors	Assurance Engineer
Assumptions	The basic (i.e. non- parameterized) system architecture shall be already defined.
Post-conditions	The architecture defined by parameters is available
Steps	1. Select the instance of the subcomponent that is going to represent an array of subcomponent instances. 2. In the property tab, edit the multiplicity attribute, that is the number of subcomponents instances of the same subcomponent. The multiplicity can be expressed as an arithmetic expression. 3. Select the port that is going to represent an array of ports of the same type. 4. In the property tab, edit the multiplicity attribute, that is the number of ports that the selected port represents. The multiplicity can be expressed as an arithmetic expression. 5. Create one or more constraints to define the connection between the parameterized parts (subcomponents and ports). 6. Create one or more constraints to define the boundaries on the number of subcomponents and ports.
Variations	None
Non-functional	None
Requirements	WP3_APL_002

Table 14. Use case “Provide a configuration constraint for a parameterized architecture”

Use Case	Provide a configuration constraint for a parameterized architecture
Functionality Description	The System component Specification Editor shall allow the user to specify a configuration constraint for a defined parameterized architecture
Actors	Assurance Engineer
Assumptions	The parameterized system architecture shall be already defined.
Post-conditions	The architecture defined by parameters is available and configured.
Steps	1. Create one or more constraints to configure the parameters used to define the multiplicity of ports and subcomponents.
Variations	None
Non-functional	None
Requirements	WP3_APL_002

3.1.2.4 Contract-based Assurance Composition Use Cases

Each component comes with a set of external-visible functional and non-functional properties; these properties must be guaranteed to hold for the component, if needed by assuming that the environment on which the component is placed behaves in a certain manner. The information about the aforementioned assumptions and guarantees is provided for a given component through the concept of **contract**, which in particular can be specified according to safety requirements previously derived. In turn, components contracts and in particular their assumptions and guarantees can be related to the assurance case specification, e.g. to support argumentation about the properties of the component/system, so to support the architecture-driven assurance approach.

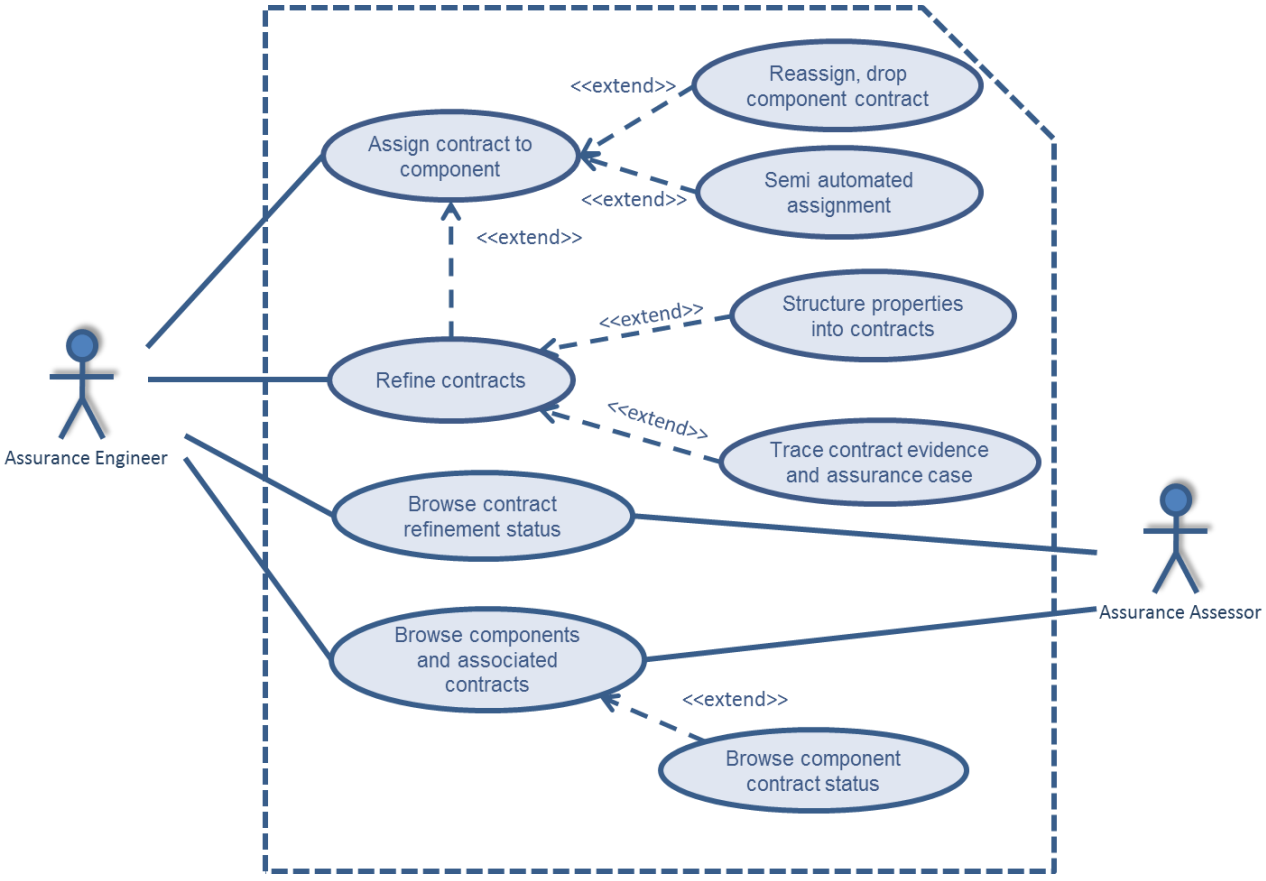


Figure 9. Use Cases for "Contract-based assurance composition" module

Table 15. Use case “Assign contracts to component”

Use Case	Assign a contract to the component
Functionality Description	The system should allow associating a contract to a component.
Actors	Assurance engineer
Assumptions	The actor has proven knowledge about modelling languages for contracts. The component is available in the model.
Post-conditions	The Contract conforms to the CACM.
Steps	1. The user selects the component model and opens it. 2. The user creates a new component contract. 3. The user assigns the contract to the component.
Variations	# Semi automated assignment of a contract to a component 1. The user selects the component model and opens it. 2. The user creates and assigns exclusively the contract to that specific component. This variant enables the content assist during the editing of the contract (see Use Case "Structure properties into contracts"). # Drop the component contract 1. The user selects a component contract 2. The user selects to delete the contract and its content # Reassign component contract 1. The user selects a component contract 2. The user updates the component to which the contract is associated and all its content is reassigned.
Non-functional	None
Requirements	WP3_CAC_009, WP3_CAC_002

Table 16. Use case "Refine component contracts"

Use Case	Refine component contract
Functionality Description	The system should allow providing contract information, the contract contents.
Actors	Assurance Engineer
Assumptions	The actor has proof knowledge about modelling languages for contracts. Contract and component are available in the model.
Post-conditions	Information provided about contract refinement conforms to the CACM.
Steps	1. The user selects the component contract and opens it. 2. The user edits, modifies or drops information about contract refinement.
Variations	None
Non-functional	None
Requirements	WP3_CAC_004

Table 17. Use case "Structure properties into contracts"

Use Case	Structure properties into contracts (assumptions/guarantees)
Functionality Description	The contract should allow to model a contract as aggregation of assumptions and to guarantee formal properties.
Actors	Assurance Engineer
Assumptions	The actor has proof knowledge about the modelling languages such as SysML, UML, CHESS profile for contracts. Properties are available in the model and are specified in a formal way.
Post-conditions	None

Steps	1. The user selects the component contract and opens it 2. The user selects the option of editing contract properties 3. The user uses the Properties view to bind the existing Formal Properties from the component as contract's assumption and guarantee.
Variations	# Automatic creation of assumptions/guarantees 1. The tool automatically creates the (empty) assumption and guarantee Formal Property for the created Contract. # Contract/property Editor with Content Assist 1. The user selects a contract component or a formal property. 2. The user edits the body of the contract component/formal property with the support of a content assist on: ports and attributes of the component, and of keywords of the current language. 3. Each error is notified by the content assist in the Editor view, in the Error view, and graphically on the current element in the Model Explorer view.
Non-functional	None
Requirements	WP3_CAC_009, WP3_CAC_003, WP3_CAC_013, WP3_SC_004

Table 18. Use case "Trace contract to evidence and assurance case"

Use Case	**Trace contract to evidence and assurance case-related information**
Functionality Description	The contract editor shall allow the user to specify the links between the contract information and the evidence and assurance case-related information.
Actors	Assurance Engineer
Assumptions	Component contracts and evidence-and assurance case-related information shall be already defined.
Post-conditions	The links conforms to the CACM specification.
Steps	1. The user selects inside the component contract, the assumption or guarantee to be linked. 2. The user selects the fragments for the assurance case to be linked. 3. The tool allows the user to create the links to the proper and evidence or assurance case-related information.
Variations	None
Non-functional	None
Requirements	WP3_VVA_001, WP3_SAM_001

Table 19. Use case "Browse components and associated contracts"

Use Case	**Brose components and associated contracts**
Functionality Description	The system should allow providing information about the modelled components and associated contracts.
Actors	Assurance Engineer
Assumptions	None
Post-conditions	None
Steps	1. The user selects the dedicated view to check the status of the components currently defined in the system architecture, together with the associated contracts.
Variations	None
Non-functional	None
Requirements	WP3_CAC_005

Table 20. Use case "Browse component contracts status"

Use Case	Browse component contracts status
Functionality Description	The contract editor shall be able to show the user the contracts associated within a component and their status (fulfilled or not).
Actors	Assurance Engineer, Assurance Assessor
Assumptions	Component contracts shall be already defined.
Post-conditions	None
Steps	1. The user selects a component and from a menu selects the show contracts option. 2. In a view, the user is able to see all the contracts associated with the selected component. 3. For each of the contracts, when selected, the user is able to see the status, if it is already fulfilled or not.
Variations	None
Non-functional	None
Requirements	WP3_CAC_012

Table 21. Use case "Browse Contracts refinement status"

Use Case	Brose Contracts refinement status
Functionality Description	The system should allow providing information about the modelled contracts refinement.
Actors	Assurance Engineer
Assumptions	The actor has proven knowledge about modelling languages for contracts. Contracts refinement is available in the model.
Post-conditions	None
Steps	1. The user selects the dedicated view to check the status of the contracts refinements along the system architecture.
Variations	None
Non-functional	None
Requirements	WP3_CAC_006

3.1.2.5 Requirements Support Use Cases

This section addresses the uses cases which are related to Requirements Support.

Table 22. Use case "Requirements formalisation"

Use Case	Requirements formalisation
Functionality Description	The AMASS platform must be able to formalise requirements into formal properties (i.e., expressions in a language with a formal semantics such as temporal logics). This enables the application of formal verification.
Actors	Assurance Engineer
Assumptions	The actor has proof knowledge about the modelling languages such as SysML, UML, CHESS profile for contracts. Requirements must be available
Post-conditions	None
Steps	1. The user selects the requirements and imports them 2. The user selects and formalises the requirement
Variations	None

Non-functional	None
Requirements	WP3_SC_004

Table 23. Use case "Analysis of requirements' semantics based on their formalisation into temporal logics"

Use Case	**Analysis of requirements' semantics based on their formalisation into temporal logics**
Functionality Description	The ARTA shall enable users to analyse the requirements to validate formal requirements/properties.
Actors	Assurance engineer
Assumptions	Requirements must be already defined and formalised into temporal logic. A tool for the analysis of requirements semantics which is properly interconnected to ARTA will complete the validation of the requirements.
Post-conditions	Requirements are verified.
Steps	1. The user requirements to be analysed 2. The user runs an external tool such as Knowledge Manager to complete the validation of those requirements. 3. The user analyses the results of the validation and identifies possible redundant or inconsistent requirements.
Variations	None
Non-functional	None
Requirements	WP3_VVA_003, WP3_SC_004

Table 24. Use case "Requirements Semantic Analysis"

Use Case	**Requirements Semantic Analysis**
Functionality Description	The ARTA shall enable users to semantically analyse requirements for logical consistency, non-redundancy, and realisability (system can be created from the requirements).
Actors	Assurance engineer
Assumptions	Requirements shall be already defined. The external V&V tools or verification servers are properly installed through the ARTA preferences page.
Post-conditions	Requirements are logically consistent, non-redundant, and realisable.
Steps	1. The user selects the contracts to be analysed 2. The user selects Validate → V&V Manager → Requirements Semantic Analysis 3. Verification server calls 4. The user sees results from the analysis in V&V Results windows and removes all defects in contracts (and corresponding requirements) and iteratively executes the Requirements Semantic Analysis till all contracts are logically consistent, non-redundant, and realisable.
Variations	# Quality analysis • The user selects and configures a set of metrics to perform a quality analysis (about correctness, consistency, completeness…). The system shows a quality report, which might include a quality evolution information.
Non-functional	None
Requirements	WP3_VVA_003, WP3_SC_004

3.1.2.6 V&V Activities

This group of use cases deals with the possibility to perform verification and validation analysis based upon the currently available system model specification, where the latter comprises components hierarchical specification, nominal and error behavioural models, and contracts associated to the components. The goal of the proposed features is to support architecture-driven assurance, so to allow the generation of evidence and to support dedicated claims available in the system assurance case.

Figure 10 depicts the use cases regarding V&V activities. The new use cases are highlighted in yellow e.g. simulation-based fault injection which pursues an early safety validation of the system.

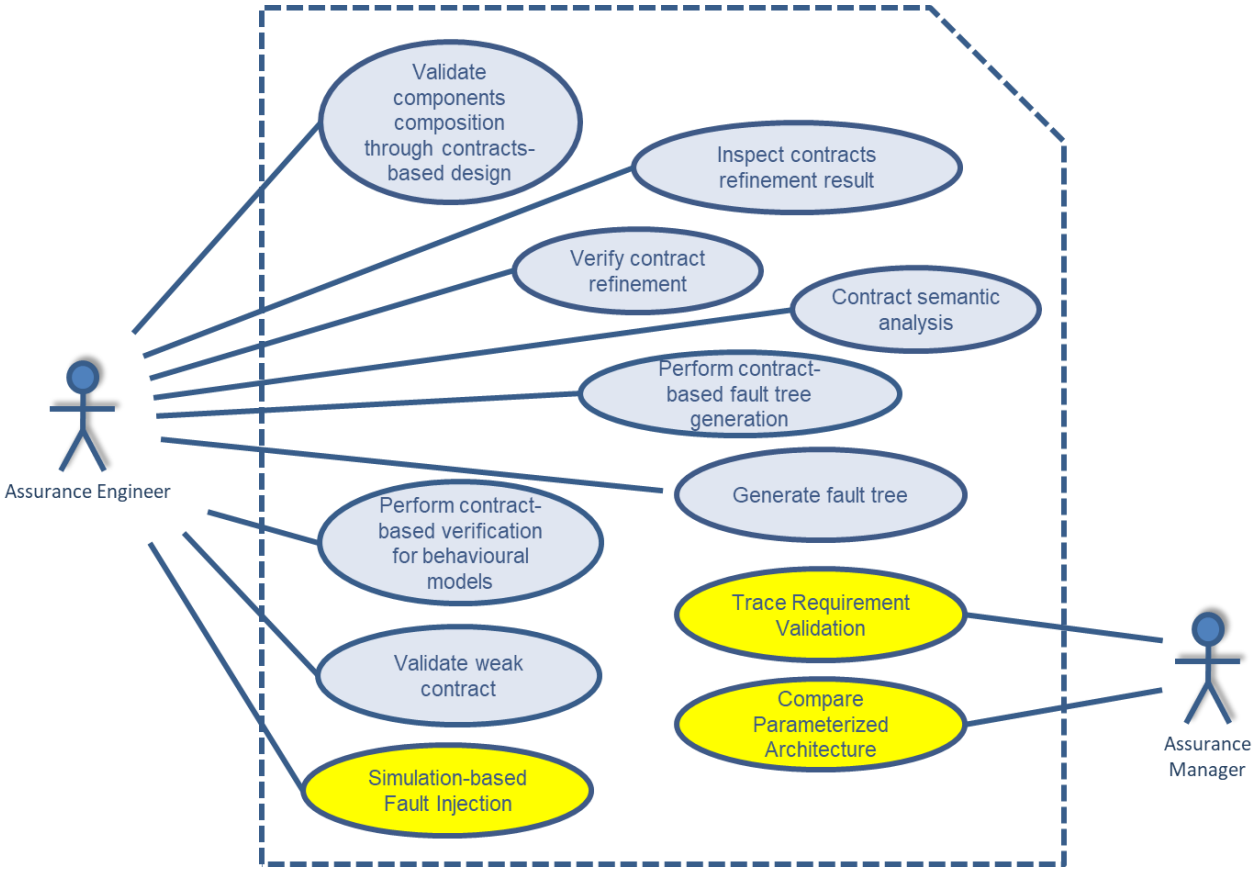


Figure 10. Use cases for V&V Activities module

Table 25. Use case "Validate components composition through contracts-based design"

Use Case	Validate components composition through contracts-based design
Functionality Description	The ARTA shall enable the validation the components composition by checking the compatibility of the contracts available for the components themselves.
Actors	Assurance engineer
Assumptions	Component contracts shall be already defined.
Post-conditions	None
Steps	1. The user selects the "Check contracts compatibility" functionality. 2. The user selects the guarantee and the assumption of two components contracts. 3. The user selects the type of check to perform, i.e. consistency, possibility, and entailment. 4. ARTA allows to perform the validation and to show the result of the verification
Variations	None

Non-functional	None
Requirements	WP3_CAC_001

Table 26. Use case "Verify contract refinement"

Use Case	**Verify contract refinement**
Functionality Description	The ARTA shall enable users to check that a given system architecture is correct with respect to the specified refinement of contracts
Actors	Assurance engineer
Assumptions	Component contracts and their refinement shall be already defined for the components to be analysed. The external tool allowing contracts refinement is properly configured through the ARTA preferences page.
Post-conditions	None
Steps	1. The user selects the Check Contracts Refinement functionality 2. The ARTA enables the verification of the system model and the extraction of the information to be given in input to the external tool 3. The ARTA allows to invoke the external tool with the collected data in input, waiting for the result. 4. The result produced by the external tool is about the contract refinement analysis result is presented by the ARTA to the user.
Variations	1.a The user invokes the Check Contract Refinement Functionality by specifying that all the weak contracts available in the model have to be considered for the analysis. #Verify contract refinement with all the weak contracts: The contract refinement analysis cannot be performed, then a report with a list of model errors is generated.
Non-functional	None
Requirements	WP3_CAC_008, WP3_VVA_007

Table 27. Use case "Contract Semantic Analysis"

Use Case	**Contract Semantic Analysis**
Functionality Description	The ARTA shall enable users to semantically analyse contracts for logical consistency, non-redundancy, and realizability (system can be created from the requirements).
Actors	Assurance engineer
Assumptions	Component contracts shall be already defined. The external V&V tools or verification servers are properly installed through the ARTA preferences page.
Post-conditions	Contracts are logically consistent, non-redundant, and realisable.
Steps	5. The user selects the contracts to be analysed 6. The user selects Validate → V&V Manager → Contract Semantic Analysis 7. Verification server calls 8. The user sees results from the analysis in V&V Results windows and removes all defects in contracts (and corresponding requirements) and iteratively executes the Contract Semantic Analysis till all contracts are logically consistent, non-redundant, and realisable.
Variations	# Quality analysis • The user selects and configures a set of metrics to perform a quality analysis (about correctness, consistency, completeness…). The system shows a quality report, which might include a quality evolution information.
Non-functional	None

Requirements	WP3_VVA_003

Table 28. Use Case "Perform contract-based validation for behavioural models"

Use Case	Perform contract-based verification of behavioural models
Functionality Description	The ARTA shall enable users to verify if the state machine satisfies the contracts.
Actors	Assurance engineer
Assumptions	Components contracts and state-machines shall be already defined for the system components to be analysed. The external tool allowing contract-based validation for behavioural models is properly configured through the ARTA preferences.
Post-conditions	None
Steps	1. The user selects the "Perform contract-based verification for behavioural models" functionality for the selected components 2. The ARTA verifies if the machine satisfies the contracts and extract the information to be given in input to the external tool 3. The ARTA invokes the configured external tool with the collected data in input, waiting for the result 4. For each component and contract, the result produced by the external tool about the contract verification against the behavioural model of the component is presented by the ARTA to the user.
Variations	2.a The generation cannot be performed, then a report with a list of model errors is generated
Non-functional	None
Requirements	WP3_CAC_011, WP3_VVA_005, WP3_VVA_007

Table 29. Use Case "Inspect contracts refinement result"

Use Case	Inspect contracts refinement result
Functionality Description	The ARTA shall enable users to have an overview of contracts refinement check result.
Actors	Assurance engineer
Assumptions	Component contracts shall be already defined and contract refinement analysis shall be already performed.
Post-conditions	None
Steps	1. The user opens the Contract Refinement View 2. The user navigates the view to check the refinement status for all the defined contracts.
Variations	None
Non-functional	None
Requirements	WP3_CAC_007, WP3_VVA_007

Table 30. Use case "Generate fault trees"

Use Case	Generate fault tree
Functionality Description	The ARTA shall enable users to generate the fault trees starting from the modelled nominal and failure behaviour.
Actors	Assurance engineer
Assumptions	Nominal and failure behaviour shall be already defined for the system components to be analysed.

	The external tool allowing fault trees generation is properly configured through the ARTA preferences.
Post-conditions	None
Steps	1. The user selects the “Generate fault tree” command responsible to perform the fault tree generation. 2. The ARTA enables the validation of the system model and the extraction of the information to be given as input to the external tool 3. The ARTA allows to invoke the external tool with the collected data in input, waiting for the result 4. The result, that is a fault-tree, is graphically shown by the ARTA to the user.
Variations	# Visualization of existing fault tree. 1. The user selects a file that represents a fault tree. 2. The fault tree is shown to the user
Non-functional	None
Requirements	WP3_VVA_010, WP3_VVA_006

Table 31. Use case “Perform contract-based fault trees generation”

Use Case	**Perform contract-based fault tree generation**
Functionality Description	The ARTA shall enable users to generate the fault trees starting from the contract-based decomposition of the system.
Actors	Assurance engineer
Assumptions	The contract-based decomposition of the system shall be already defined for the system components to be analysed. The external tool allowing fault tree generation is properly configured through the ARTA preferences.
Post-conditions	None
Steps	1. The user selects the “Generate contract-based fault tree” command 2. The ARTA allows to validate the system model and extract the information to be given in input to the external tool 3. The ARTA allows to invoke the external tool with the collected data in input, waiting for the result 4. The result, that is a fault-tree, is graphically shown by the ARTA to the user.
Variations	None
Non-functional	None
Requirements	WP3_CAC_008, WP3_VVA_006, WP3_VVA_010

Table 32. Use case “Validate weak contracts”

Use Case	**Validate weak contracts**
Functionality Description	The ARTA shall enable users to validate weak contract assumptions and identify whether the weak contract is relevant in the given context.
Actors	Assurance engineer
Assumptions	Component contracts and their refinement shall be already defined for the components to be analysed. The external tool allowing contracts refinement is properly configured through the ARTA preferences page. The contract specifications comply with the discrete time model.
Post-conditions	None

Steps	1. The user invokes the Validate Weak Contracts functionality by specifying that all weak contracts have to be considered for the analysis. 2. The ARTA allows to validate the system model and extract the information to be given in input to the external tool. 3. The ARTA invokes the external tool with the collected data in input, waiting for the result. 4. The result produced by the external tool summarising the validity of the contract assumptions is saved and presented to the user by the ARTA.
Variations	2.a The contract refinement analysis cannot be performed, then a report with a list of model errors is generated.
Non-functional	None
Requirements	WP3_CAC_008, WP3_CAC_012, WP3_VVA_007

Table 33. Use case "Compare parameterized architecture"

Use Case	**Compare Parameterized Architecture**
Functionality Description	The System Component Specification Editor shall allow the user to compare parameterized architecture with respect to a number of properties based on formal verification and analysis
Actors	Assurance Engineer
Assumptions	A set of configurations, with a parameterized architecture or with a set of architectural models, has been defined
Post-conditions	None
Steps	1. The user invokes the command “Compare System Architectures” 2. The user selects one or more architectures to analyse. 3. In case of parameterized architectures, the user selects the set of configurations that are going to be instantiated (see Table 14). 4. The user selects which are the aspects that are going to be compared. 5. A file, that is graphically interpreted in a dedicated view, contains for each architecture and for each aspect, qualitative and quantitative results.
Variations	None
Non-functional	None
Requirements	WP3_APL_002

Table 34. Use case "Trace requirement validation"

Use Case	**Trace Requirement validation**
Functionality Description	The system editor shall allow the user to trace requirements to the V&V analysis used to test them
Actors	Assurance Engineer
Assumptions	The requirement to be traced is available in the system model. An analysis context has been used to store the information needed to perform the V&V analysis related to the requirement to be traced.
Post-conditions	
Steps	1. The user selects the requirements and the analysis context to be traced 2. The user creates a traceability links between the selected entities.
Variations	None
Non-functional	None
Requirements	WP3_VVA_004

Table 35. Use case "Simulation-based fault injection"

Use Case	Simulation-based Fault Injection
Functionality Description	The system editor shall allow the user to configure, run and analyse simulation-based fault injection simulations
Actors	Assurance Engineer (V&V Engineer), System Engineer, Assurance Manager
Assumptions	The system model is extended with faulty information
Post-conditions	The safety concept has been early validated
Steps	1. The user annotates the system model with faulty information 2. The user configures the rest of the needed information to create the fault list (e.g. fault target, fault injection time trigger) 3. The user runs the fault injection simulations 4. Simulation Results are analysed and proper measures are taken. If the inserted faults are not detected or the measures are not sufficient, the user redesigns the safety concept/safety mechanisms.
Variations	#Integration of component contracts- Simulation-based fault injection simulations and generation of monitors for the early safety V&V of system design and automation of safety analysis.
Non-functional	None
Requirements	WP3_VVA_011

Table 36. Use case "Monitoring for Functional Verification"

Use Case	Monitoring for Functional Verification
Functionality Description	The ARTA shall allow the user to generate monitors from contracts that will be used to automatically observe simulation traces and check whether they satisfy the contract
Actors	Assurance Engineer (V&V Engineer)
Assumptions	Component contracts shall be already defined
Post-conditions	None
Steps	1. The user simulates the system model and records the simulation traces 2. The user runs the monitoring tool on the recorded simulation traces, checking whether they satisfy the contracts 3. The user uses the trace diagnostics information from the monitoring tool to understand the reasons of the detected faults, if any, and use it to debug the model or the contract.
Variations	#falsification-based testing – Combine the monitoring (quantitative) verdicts with search-based techniques to efficiently generate new tests that will guide the system model towards contract violation.
Non-functional	Security policies as contracts
Requirements	WP3_VVA_011, WP3_VVA_005

Table 37. Use case "Generate documentation from the system model"

Use Case	Generate documentation from the system model
Functionality Description	The system report shall provide the capability for generating a report about system components definition and analyses results
Actors	Assurance Manager
Assumptions	The system architecture shall be already defined.

Post-conditions	One or more files describing the system architecture are generated.
Steps	1. The user selects the system components or one sub-component. 2. The user selects the generate report menu command 3. The system editor creates the report in a dedicated folder. In case of errors, the system notifies to the user such errors.
Variations	None
Non-functional	None
Requirements	WP3_VVA_007

3.1.3 Multi-concern Assurance Uses Cases (*)

This section addresses the uses cases which are related to STO2 Multi-concern Assurance.

3.1.3.1 Assurance Case Specification Use Cases

3.1.3.1.1 Assurance Case Edition Use Cases

This group of use cases manages argumentation information in a modular fashion. It also includes mechanisms to support assurance patterns management.

Assurance cases are a structured form of an argument that specifies convincing justification that a system is adequately dependable for a given application in a given environment. Assurance cases are modelled as connections between claims and their evidence. Ideally, they should provide an argument for every possible scenario with regards to dependability of a system, and give proof through justifications and evidence that a system is indeed dependable (safe, secure, reliable, and the like).

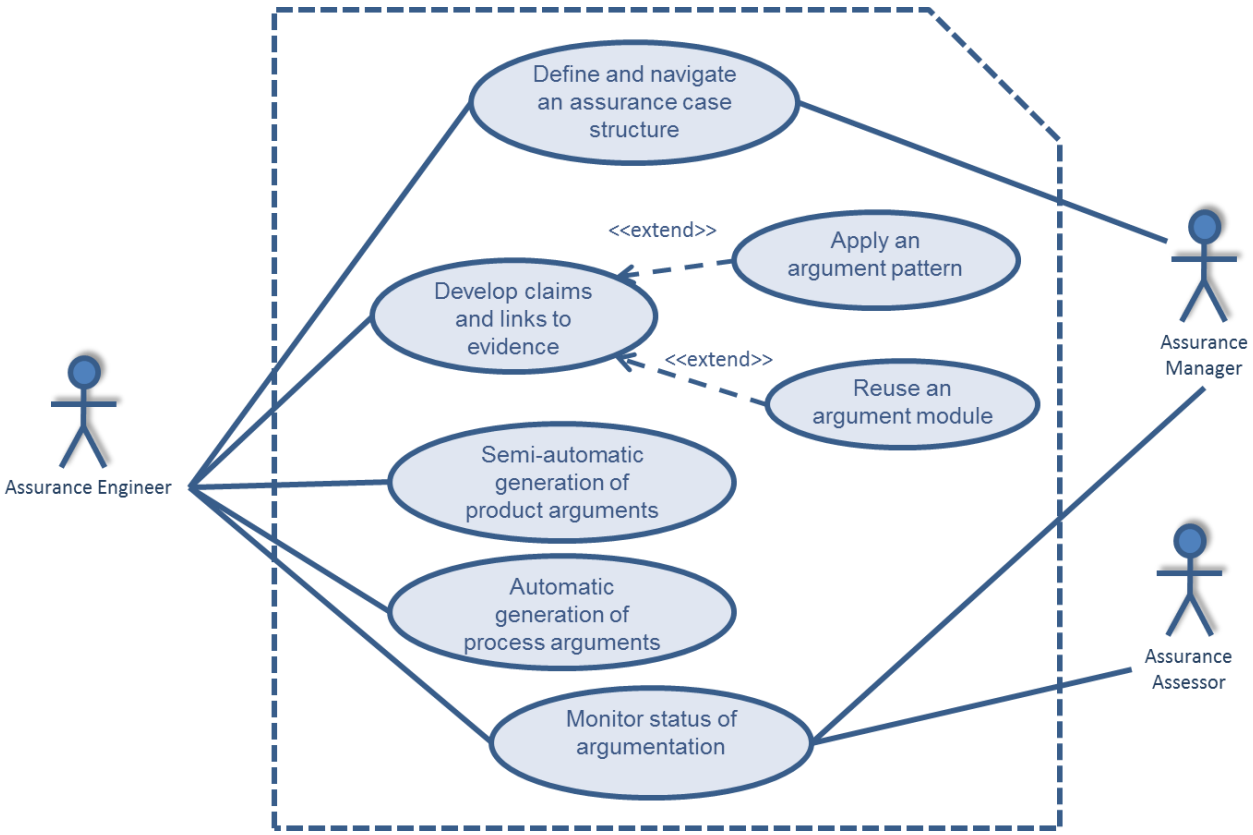


Figure 11. Use Cases for "Assurance Case Specification" module

Full automation and formalisation of Assurance Cases is unlikely to be feasible (or desirable). Therefore, human review and judgement is ultimately required to assess whether a system is "acceptably dependable", and this stems from an understanding of the argument and compatibility of evidence. Nevertheless, AMASS WP4 must provide some structured and formalized process support to the tasks of assurance case validation and composition. However, the AMASS tools may provide the (semi-automatic) generation of argument fragments based on information specified in other modules (e.g. system component specification and process compliance management).

Writing and validating assurance cases is difficult because they tend to be huge and complex, and they require domain specific knowledge of target systems. To ease these difficulties, the AMASS tools must support the collection of assurance case patterns in libraries so that they are available for reuse, similarly to design patterns in object-oriented languages.

Table 38. Use case "Define and navigate an assurance case structure"

Use Case	Define and navigate an assurance case structure
Description	The actor aims to use assurance case modules as basic composable specification. This use case corresponds to the scenario to define an integrated and structured assurance case where the actor can navigate through the structure.
Actors	Assurance Manager, Assurance Engineer
Assumptions	We have different levels of argumentation abstraction. Assurance case modules can be composed together if their Goals match and their Context is compatible.
Pre-conditions	None
Post-conditions	The modular assurance structure for a given project has been detailed.
Steps	In an argumentation diagram the user will: • Define the appropriate granularity by using argument modules to encapsulate arguments • Inside each argument module include appropriate arguments taking into account: hazard mitigation, requirements, integration, …
Variations	# Apply an argument pattern-based structure Check the feasibility of reusing previously created argument packages.
Non-functional Requirements	None
	WP4_ACS_001, WP4_ACS_002

Table 39. Use case "Develop Claims and Links to Evidence"

Use Case	Develop claims and links to evidence
Description	The system should help users to identify and define the most appropriate arguments and evidence to support their goals.
Actors	Assurance Engineer
Assumptions	The actor uses guidelines and support to apply the best practices to develop the statements of argument structures.
Pre-conditions	The current argumentation module has been created. The pieces of evidence addressed by the current project have been established.
Post-conditions	The current Argumentation Module is completely defined.
Steps	For every argument module: 1. Specify manually the claims set 2. Provide stated and valid assumptions applied to the claims 3. Map to the available pieces of evidence that support the claims

	4. Specify contextual information to define or constraint the scope over which the arguments are assumed to be valid 5. When required, map claims (away goals) to the external claims (public goals) that support to (in other argument modules) 6. Assess the confidence of every composed argument
Variations	# Reuse and argument module (Import additional pieces of argumentation set) from an external file # Mark claims (public goals) that can be offered to others argument modules # Select an argument pattern to substantiate or address particular claims
Non-functional	None
Requirements	WP4_ACS_001, WP4_ACS_005

Table 40. Use case "Apply an argument pattern"

Use Case	**Apply an argument pattern**
Description	This use case corresponds to the capability to instantiate an argument pattern (e.g., concerning safety and security) selected from the list of stored patterns.
Actors	Assurance Engineer
Assumptions	Assurance patterns have been specified and stored following the general procedure to create assurance cases.
Pre-conditions	The assurance argumentation is under edition.
Post-conditions	Changes are registered
Steps	1. Library of patterns is available to be used in a specific assurance case model 2. Drag and drop argument pattern into the desired diagram of assurance case 3. Pattern parameters must be defined by the user 4. Provide guidelines to use and instantiate argument pattern (e.g., concerning safety and security) presented in the actual assurance case.
Variations	None
Non-functional	None
Requirements	WP4_ACS_003

Table 41. Use case "Reuse an argument pattern"

Use Case	**Reuse an argument module**
Description	This use case corresponds to the capability to reuse an argument module (e.g., concerning an element of the system) selected from the list of stored modules.
Actors	Assurance Engineer
Assumptions	Assurance modules have been specified and stored following the general procedure to create assurance cases.
Pre-conditions	The assurance argumentation is under edition.
Post-conditions	Changes are registered
Steps	1. Library of modules is available to be used in a specific assurance case model 2. Drag and drop argument module into the desired diagram of assurance case
Variations	None
Non-functional	None
Requirements	WP4_ACS_001, WP4_ACS_002, WP4_ACS_003

Table 42. Use case "Semi-automatic generation of product arguments"

Use Case	**Semi-automatic generation of product arguments**

Functionality Description	The AMASS tools shall enable semi-automatic generation of product-based argument-fragments
Actors	Assurance Engineer
Assumptions	Strong and weak component contracts shall be already defined and associated with claims, context statements and evidence artefacts. The weak contracts shall be either selected for usage in the given context, or all weak contract assumptions shall be validated. The contract refinement analysis shall be already performed, either for the selected contracts, or for all the weak contracts.
Post-conditions	None
Steps	1. The user selects the “Generate argumentation fragments” functionality 2. The user selects either new or existing assurance project as the destination for the argument-fragments 3. The ARTA validates the system model and extracts the information needed for the argument-fragment generation for each component 4. The ARTA generates the corresponding argument-fragments, and notifies the user of their location.
Variations	None
Non-functional	None
Requirements	WP6_PPA_002

Table 43. Use case “Automatic generation of process arguments”

Use Case	**Automatic generation of process arguments**
Functionality Description	The AMASS tools shall enable automatic generation of process-based argument-fragments.
Actors	Safety Engineer, Security Engineer
Assumptions	A process model has been specified.
Post-conditions	None
Steps	1. The user selects the “Generate argumentation fragments” functionality. 2. The user selects either new or existing assurance project as the destination for the argument-fragments. 3. The information needed for the argument-fragment generation is extracted from the process model. 4. The corresponding argument-fragments are generated; the location is notified to the user.
Variations	None
Non-functional	None
Requirements	WP6_PPA_003

Table 44. Use case “Monitor status of argumentation”

Use Case	**Monitor status of argumentation**
Description	The system shall allow a user to query assurance case progress and particular aspects such as undeveloped goals.
Actors	Assurance Manager, Assurance Engineer, Assurance Assessor.
Assumptions	None
Pre-conditions	An assurance case is being maintained.
Post-conditions	None

Steps	For every Argument diagram: 1. Claims with no supported evidence or decomposed are identified 2. The overall assurance case completion is presented.
Variations	None
Non-functional	None
Requirements	WP4_ACS_011

3.1.3.2 Dependability Modelling Use cases

This group of use cases refers to structure multi-concern assurance argumentation in a way that allows easily understanding interactions and supporting the maintainability of the assurance case and the system. The use cases related to "Navigate an assurance case structure" and "Develop claims and links to evidence" haven been explained previously in tables Table 38 and Table 39 respectively.

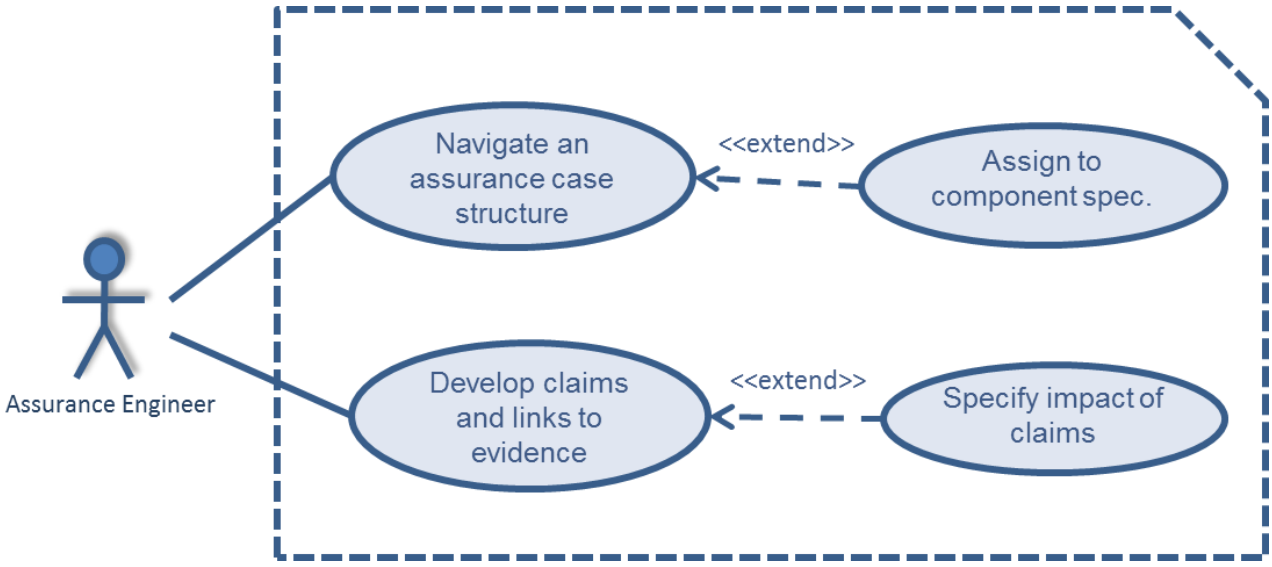


Figure 12. Use Cases for "Dependability Modelling" module

Table 45. Use case "Assign to component specification"

Use Case	**Assign to component specification**
Description	The actor aims to use assurance case modules as basic composable specification in combination with the system component architecture. This use case corresponds to the scenario to define argument modules encapsulating the assurance cases corresponding to the components from the system component architecture where the actor can navigate through the structure.
Actors	Assurance Engineer
Assumptions	The argumentation in the assurance case has been developed in correlation with the system architecture.
Pre-conditions	There is a system architecture definition.
Post-conditions	The system architecture and the assurance case specifications are correlated.
Steps	1. Open the assurance case structure view 2. Select one of the argument modules 3. Connect the argument module with the system architecture 4. Select the component design from the list of components included in the system architecture specification design to be connected.
Variations	None
Non-functional	None

Requirements	WP4_ACS_002, WP4_ACS_008, WP4_ACS_010

Table 46. Use case "Specify impact of claims"

Use Case	Specify impact of claims
Description	The system should help users to identify and argue about some assertions made in the context of one concern (such as security) and the impact in other concerns (such as safety)
Actors	Assurance Engineer
Assumptions	The actor uses guidelines and support to apply the best practices to develop the statements of argument structures.
Pre-conditions	The current argumentation module has been created. The pieces of evidence addressed by the current project have been established.
Post-conditions	The current Argumentation Module is under edition.
Steps	For each of the claims referring to a specific concern: 1. Analyse the impact of another claim 2. Specify the possible impact relationships described in D4.2 [11] and D4.3 [15]: a) Dependency relationship, b) Conflicting relationship or c) Supporting relationship
Variations	None
Non-functional	None
Requirements	WP4_DAM_001, WP4_DAM_002

3.1.3.3 Contract-based multi-concern assurance Use Cases

The use cases related to contract-based multi-concern assurance refer to the possibility of identifying the different assurance concerns (e.g. safety, security, performance…) and relating them with the existing contracts.

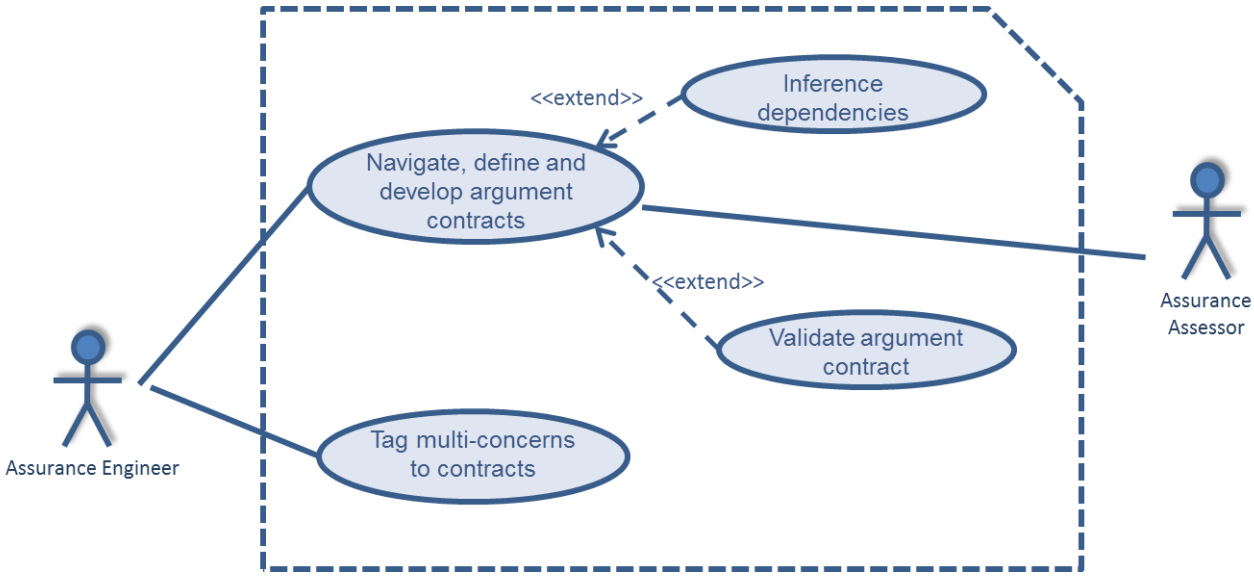


Figure 13. Use Cases for "Contract-based multi-concern assurance" module

Table 47. Use case "Navigate, define and develop argument contracts"

Use Case	Navigate, define and develop argument contracts

Description	The actor aims to take advantage of the compositional argumentation approach. Argument modules can encapsulate arguments and include references in their content arguments developed in other argument modules. In the argument contracts these dependencies should be included and argue about the composition adequacy.
Actors	Assurance Engineer
Assumptions	None
Pre-conditions	The arguments in the assurance case are completely defined
Post-conditions	The argument contract is created
Steps	In the argumentation diagram: 1. Indicate a new Argument Contract figure 2. Connect the Argument contract with at least two or more argument modules. 3. Edit a new argument diagram with the content of the argument contract 4. In the location property of the Argument contract figure, indicate the address to where the content of the argument is.
Variations	#The diagram of the argument contract is created if the location is empty
Non-functional	None
Requirements	WP4_ACS_001, WP4_ACS_002, WP4_ACS_010

Table 48. Use case “Inference dependencies”

Use Case	**Inference dependencies**
Description	The actor aims to take advantage of the compositional argumentation approach. Argument modules can include references in their content arguments to claims and evidence developed in other argument modules. In the argument contracts these dependencies should be included and if not, inferenced.
Actors	Assurance Engineer
Assumptions	None
Pre-conditions	The arguments in the assurance case are completely defined
Post-conditions	The argument contract includes all dependencies
Steps	For each argument contract: 1. Indicate at least one of the argument modules supported by the contract 2. Select to inference dependencies 3. The references to the dependencies appear in the contract together with references to other argument modules 4. Repeat the action to inference dependencies of the new argument modules included 5. Develop arguments about the dependencies
Variations	None
Non-functional	None
Requirements	WP4_ACS_010

Table 49. Use case “Validate argument contract”

Use Case	**Validate argument contract**
Description	The actor aims to take advantage of the compositional argumentation approach. Argument contracts need to validate that all dependencies are supported by evidence
Actors	Assurance Engineer
Assumptions	None
Pre-conditions	The arguments in the assurance case are completely defined and all inferences are included in the argument contract

Post-conditions	The argument contracts include all dependencies and are validated
Steps	For each argument contract: 1. Validate the dependencies 2. Not valid dependencies are highlighted 3. User can provide valid evidence to support dependencies
Variations	None
Non-functional	None
Requirements	WP4_ACS_012

Table 50. Use case "Tag multi-concern to contracts"

Use Case	**Tag multi-concerns to contracts**
Description	The system should allow the user to identify and highlight the contracts associated with a specific concern
Actors	Assurance Engineer
Assumptions	The concerns are already defined and stored as properties
Pre-conditions	The contracts are already defined
Post-conditions	None
Steps	1. The user selects an existing contract 2. The user associate selected contract to a property/concern
Variations	A contract can be associated with different properties
Non-functional	There should be a mechanism to tag contracts and identify the concerns
Requirements	WP4_DAM_001

3.1.3.4 System Dependability Co-Analysis/Assessment Use cases

This module comprises functionalities for performing safety and security analyses and for taking care of influences between evidence (GSN solutions [16]) supporting [sub-]goals for one attribute and [sub-]goals for another one. The different quality attributes may be partially conflicting so that usually iterative processes are required.

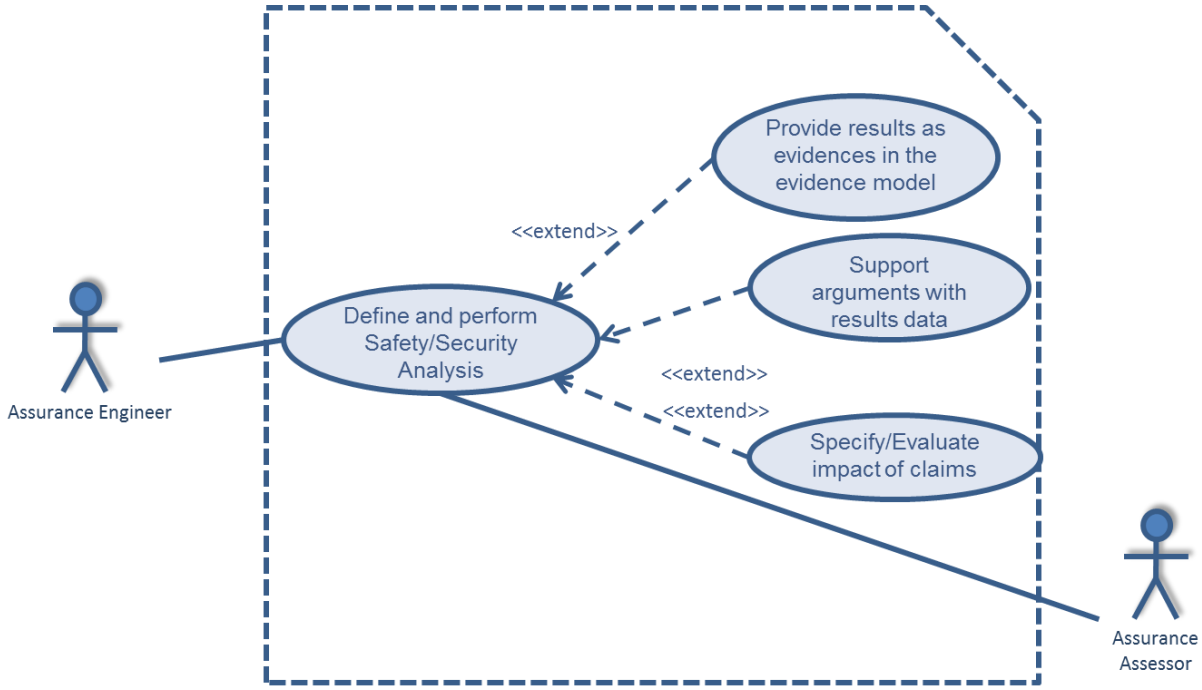


Figure 14. Use Cases for "System Dependability Co-Analysis/Assessment" module

Table 51. Use case "Define and Perform Safety/Security Analysis"

Use Case	Define and Perform Safety/Security Analysis
Description	The actor selects a method and an appropriate tool for Safety and Security Analysis. He selects an appropriate level of design/architecture for the desired granularity of the analysis. For each item of the [sub-]system/component he identifies failure and threat modes, possible causes and vulnerabilities, detection possibilities, and existing mitigation measures.
Actors	Assurance Engineer
Assumptions	The Assurance Engineer has good knowledge about safety and security analyses to make the decisions and perform the manual or monitor the partly automated analysis. The actor has good knowledge of the system properties at least on the level on which the analyses are to be conducted.
Pre-conditions	System/component definition fully specified at the planned analysis level. System/component usage environment defined.
Post-conditions	Assets to be protected identified, All conceivable failure and threat modes identified, All possible causes and vulnerabilities identified, Possibility to detect the failures/attacks assessed, Mitigation measures already in place identified.
Steps	1. Identify assets to be protected 2. Select appropriate method and tool 3. Decide on which level to analyse the [Sub-]System/Component 4. Identify for each item all conceivable failure and threat modes with all possible causes and vulnerabilities and assess possibility to detect the failures/attacks 5. Identify mitigation measures already in place
Variations	• Method perspective: FMVEA, SAHARA and – more detailed – FTA&ATA Degree of automation perspective: The Actor may rely on fault and threat mode databases and have the analysis partly generated by the tool
Non-functional	None
Requirements	WP4_SDCA_003

Table 52. Use case "Provide results as evidence"

Use Case	Provide results as evidence
Description	The actor performs the assurance steps (tests, analyses) as defined in use case "Support arguments with result data", judges whether the result is pass or fail, and sets the evidence (GSN solution) in the assurance case argument according to the result.
Actors	Assurance Engineer
Assumptions	• In case of manual assurance: Actor can operate the involved tools as the analyses need the iteration with the user (e.g. electromagnetic compatibility, HIL test …). • In case of automated assurance: Actor knows how to apply the assurance methods and the pass/fail.
Pre-conditions	System baseline for the assurance case fixed Assurance argumentation for the baseline finalized Support of arguments with evidence defined (e.g. V-Plans in WEFACT) Artefacts under assurance, assurance tool[s], if applicable test cases, tool/script for comparing the expected result with the actual one (pass/fail decision) in place and

	ready for use
Post-conditions	• In case of meeting the pass criteria: evidence in place • In case of failure: evidence not in place
Steps	• In case of automated assurance: 1. start the automated assurance activity (e.g. the WEFACT V&V-Activities) • In case of manual assurance: 1. perform the (manual) assurance step (e.g. the respective test cases) 2. verify whether the results meet the expectations In case of OK set the respective evidence (GSN solution) in the multi-concern argument to fulfilled.
Variations	• Process may be performed manually or workflow tool based • Tools may be testing or analysis tools • Result (pass/fail) may be assessed manually or by a tool/script.
Non-functional Requirements	None
	WP5_EM_004

Table 53. Use case "Support arguments with result data"

Use Case	**Support arguments with result data**
Description	The actor selects an appropriate method and tool which can generate the evidence to support the goal under consideration according to the strategy defined with the argument. He defines the assurance activity to provide the above-mentioned evidence and associates the respective argument with the assurance activity result
Actors	Assurance Engineer
Assumptions	The actor has good knowledge of GSN, of assurance methods and respective tools
Pre-conditions	Argument with goals and sub-goals as well as strategies and context defined. Strategy for proofing the sub-goal under consideration defined.
Post-conditions	Process for generating the evidence including assurance tool, repository with actual results as well as expected results defined.
Steps	1. Select method and tool for generating the evidence in compliance with the strategy defined for the sub-goal under consideration. 2. Define test data if applicable (because assurance activity might also be an analysis). 3. Define the expected result for deciding whether the evidence is valid. 4. Associate the assurance case argument sub-goal with the result (pass/fail) as the GSN solution.
Variations	Manual definition of the evidence generation process. Definition of automatic evidence generation steps with a tool, e.g. definition of V-Plans in WEFACT.
Non-functional Requirements	None
	WP5_EM_003

Table 54. Use case "Specify evaluate impact of claims"

Use Case	**Specify/Evaluate impact of claims**
Description	The actor defines which architecture-related solutions for specific claims or goals (e.g. using a design pattern to improve one of the multiple quality attributes) influence the achievement of (another) one of the multiple quality attributes and in which way (supporting/dependency/conflicting) it influences it.
Actors	Assurance Engineer

Assumptions	Actor has very good knowledge about multi-concern engineering.
Pre-conditions	[Sub-]System structure is defined. Assurance argument including the first version of solutions is defined except relations between different attributes. This first version includes attribute-specific architectural or design-related mitigation measures like e.g. safety or security patterns, which – for the second "round" of mitigation measures – will require further measures.
Post-conditions	Relations between goals related to one quality attribute and solutions related to another quality attribute identified.
Steps	1. Analyse each architectural and/or design based solution of the first assurance argument version which is targeted at one of the multiple quality attributes. 2. Assess whether it has an impact on another quality attributes and specify the (supporting/dependency/conflicting) relation in the assurance case. 3. According to the relation, evaluate the impact on other [sub-]goals in this assurance case argument version as a preparation for further architecture/design modifications.
Variations	None
Non-functional	None
Requirements	WP4_DAM_001

3.1.4 Seamless Interoperability Use Cases

3.1.4.1 Evidence Management Use Cases

The use cases for Evidence Management deal with those basic aspects related to the specification of the information of the actual artefacts that can be or are used as assurance evidence in an assurance project. The artefacts can have specific properties (e.g. the result of a test case) and be stored in external data sources as data bases and by using external tools (e.g. DOORS for a requirement). All these aspects are managed through an artefact's lifecycle, which corresponds to the changes to an artefact (e.g. creation and modifications) and can include evaluations (e.g. about the completeness of a document). Process information about the execution of an assurance process can also be associated to artefacts, such as the technique used for the creation of an artefact or the participant (i.e. person) that is the owner of a system specification. Finally, artefacts can be related between them (e.g. the test case that validates a requirement), and when an artefact changes, the change could affect other artefacts and thus a change impact analysis is usually necessary. These two features are supported by the Assurance Traceability Use Cases.

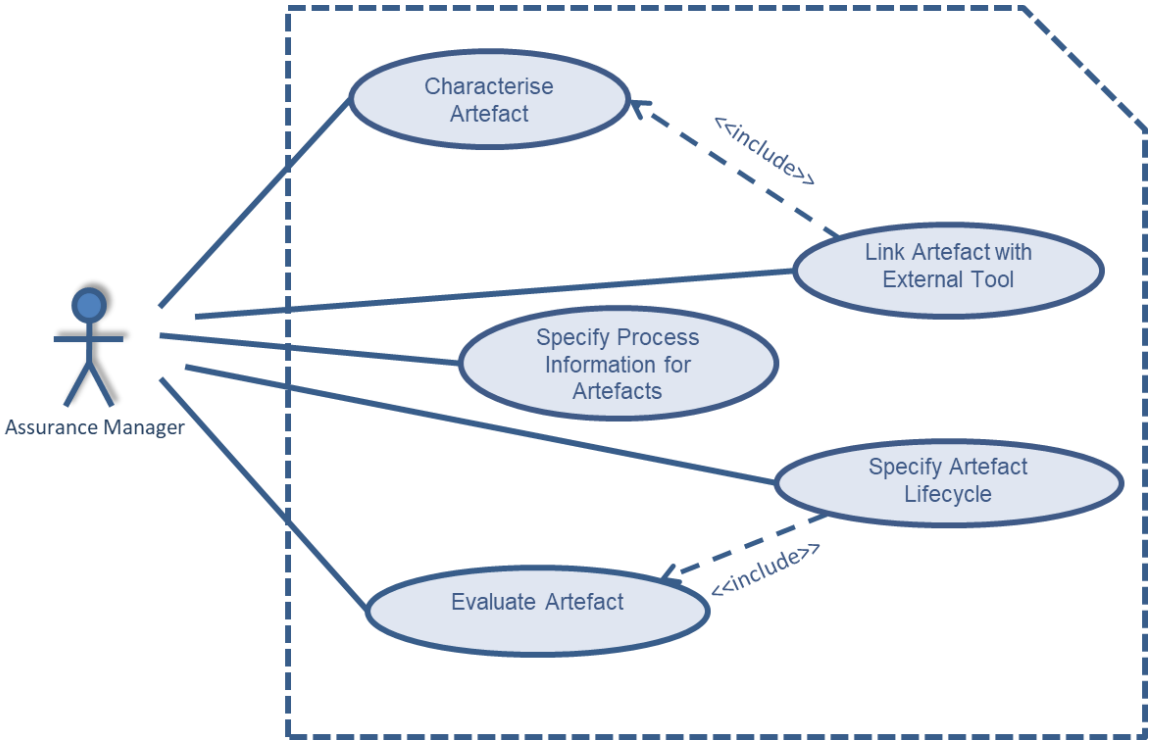


Figure 15. Use Cases for "Evidence Management" module

Table 55. Use case “Characterise Artefact”

Use Case	Characterise Artefact
Functionality Description	The AMASS Platform shall allow an Assurance Manager to specify the characteristics of assurance evidence.
Actors	Assurance Manager
Assumptions	An Artefact Definition has been created
Post-conditions	The characterization of the Artefact conforms to the CACM. The characteristics of the Artefact are shown.
Steps	1. The Assurance Manager creates an Artefact for an Artefact Definition. 2. The Assurance Manager specifies the information of the Artefact.
Variations	# The Assurance Manager adds Artefact Properties # The Assurance Manager adds sub-artefacts to the Artefact # The Assurance Manager indicates the precedent version of the Artefact # The Assurance Manager executes ‘Link Artefact with External Tool’
Non-functional	None
Requirements	WP5_EM_001, WP5_EM_007, WP5_DM_005

Table 56. Use case “Link Artefact with External Tool”

Use Case	Link Artefact with External Tool
Functionality Description	The AMASS Platform shall: (1) be able to import evidence information; (2) be able to export evidence information; (3) allow an Assurance Manager to indicate the location of the resource that an evidence artefact represents in the system. When indicating the location of the resource that an evidence artefact represents in the system, the AMASS Platform shall allow an Assurance Manager to select a part of the resource (e.g. a section inside a document or a component model file within a large system model).

Actors	Assurance Manager
Assumptions	An Artefact has been created
Post-conditions	The link with the external tool is stored in the AMASS Platform
Steps	1. The Assurance Manager selects an Artefact 2. The Assurance Manager adds a Resource to the Artefact 3. The Assurance Manager specifies the information about an External Tool in the Resource
Variations	# The Assurance Manager selects a part of the Resource # The AMASS Platform retrieves data from the external tool # The AMASS Platform exports data to the external tool
Non-functional	The AMASS Platform will connect to the external tool in less than 2 seconds
Requirements	WP5_EM_005, WP5_EM_006, WP5_EM_014, WP5_EM_015, WP5_EM_016, WP5_DM_006

Table 57. Use case "Specify Artefact Lifecycle"

Use Case	**Specify Artefact Lifecycle**
Functionality Description	The AMASS Platform shall allow an Assurance Manager to specify the events that have occurred during the lifecycle of an evidence artefact.
Actors	Assurance Manager
Assumptions	An Artefact has been created
Post-conditions	The information about the Artefact Lifecycle conforms to the CACM. The Artefact Lifecycle is shown.
Steps	1. The Assurance Manager selects an Artefact 2. The Assurance Manager adds an Artefact Event to the Managed Artefact 3. The Assurance Manager indicates the Event Kind of the Artefact Event
Variations	# When an Artefact is linked with an external tool, the lifecycle of the Artefact could be retrieved from the external tool (e.g., the modifications events of the Artefact could be determined from a SVN log). # The Assurance Manager executes 'Evaluate Artefact'.
Non-functional	None
Requirements	WP5_EM_010

Table 58. Use case "Evaluate Artefact"

Use Case	**Evaluate Artefact**
Functionality Description	The AMASS Platform shall allow an Assurance Manager to specify information about the results from evaluating an evidence artefact.
Actors	Assurance Manager
Assumptions	An Artefact has been created
Post-conditions	The evaluation information is shown
Steps	1. The Assurance Manager selects an Artefact 2. The Assurance Manager adds an Artefact Evaluation to the Artefact 3. The Assurance Manager specifies the information of the Artefact Evaluation.
Variations	# The Assurance Manager associates the Artefact Evaluation with an Artefact Event
Non-functional	None
Requirements	WP5_EM_004

Table 59. Use case "Specify Process Information for Artefacts"

Use Case	Specify Process Information for Artefacts
Functionality Description	The AMASS Platform shall allow an Assurance Manager to link evidence artefacts with other assurance assets.
Actors	Assurance Manager
Assumptions	Several Artefacts have been created.
Post-conditions	The Process Information will conform to the CACM. The Process information is shown.
Steps	1. The Assurance Manager creates an Activity 2. The Assurance Manager specifies the information of the Activity 3. The Assurance Manager indicates the Artefacts that correspond to the input and to the output of the Activity 4. The Assurance Manager creates a Participant 5. The Assurance Manager specifies the information of the Participant 6. The Assurance Manager associates an Artefact to the Participant 7. The Assurance Manager associates an Activity to the Participant 8. The Assurance Manager creates a Technique 9. The Assurance Manager specifies the information of Used Technique 10. The Assurance Manager assigns Artefacts and Activities to the Technique.
Variations	# The Assurance Manager adds a sub-activity to the Activity. # The Assurance Manager indicates the Activities that precede and succeed another Activity.
Non-functional	None
Requirements	WP5_EM_013

3.1.4.1.1 Tool integration Use Cases

This functionality deals with the two main steps for tool integration in the AMASS Tool Platform. First, it is necessary to configure the connections that will enable the integration and thus the data exchange. Second, toolchains can be defined to specify how a series of tools will be linked as a whole for a given CPS engineering or assurance process.

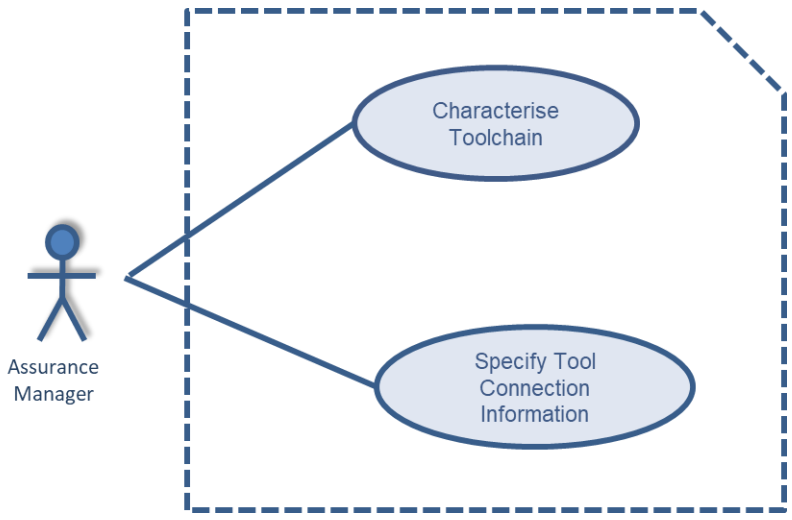


Figure 16. Use Cases for "Tool Integration" module

Table 60. Use case "Characterise Toolchain"

Use Case	Characterise Toolchain
Description	The AMASS Tool Platform shall support the specification, configuration, and deployment of tool chains for CPS assurance and certification on a single environment. In addition, the Platform shall automatically collect data from external tools, be able to automatically export data to external tools, provide exchange data in non-proprietary formats, allow users to create and enter data only once, support standard mechanisms for tool interoperability, and provide extended means to standard mechanisms for tool interoperability.
Actors	Assurance Manager
Assumptions	The tools that are going to be included in the toolchain can exchange information with the AMASS Tool Platform
Pre-conditions	Tool information is available in the AMASS Tool Platform
Post-conditions	The toolchain information is available in the AMASS Tool Platform
Steps	1. The Assurance Manager selects the tools that will be part of the toolchain 2. The Assurance Manager specifies the interactions between the tools 3. The Assurance Manager specifies the necessary information to enable the toolchain 4. The Assurance Manager is informed about the success of toolchain connection
Variations	# The Assurance Manager will be notified of the specified interactions between tools that are not supported
Non-functional	# Data will be exchanged between tools in non-proprietary formats # Data will be exchanged between tools with standard mechanisms
Requirements	WP5_TI_001, WP5_TI_002, WP5_TI_003, WP5_TI_011, WP5_TI_012, WP5_TI_013, WP5_TI_017, WP5_TI_018

Table 61. Use case "Specify Tool Connection Information"

Use Case	Specify Tool Connection Information
Description	The AMASS Tool Platform shall be able to interoperate with a wide range of tools for CPS engineering: for system analysis, system specification, V&V, version management, quality management, etc., and MS Office tools. The ultimate goal is that the Platform can interoperate with some tool in all CPS lifecycle phases. To this end, the AMASS Tool Platform shall further support data and tool integration in client-server architectures, allow clients to ask for a server's services and to discover servers, and allow continuous performance monitoring of the servers.
Actors	Assurance Manager
Assumptions	There exists some means for integration of the tool and the AMASS Tool Platform, and the means is available
Pre-conditions	None
Post-conditions	The tool connection information is available in the AMASS Tool Platform
Steps	1. The Assurance Manager creates a new tool connection 2. The Assurance Manager specifies the required information to the tool connection 3. The Assurance Manager is provided information about the success of tool connection
Variations	# The required tool connection information can vary among tools, and it can include user, password, and a URL # Alternatives can be offered for connection with a tool, e.g. ad-hoc connection or OSLC-based

Non-functional	# Tool connection will be possible in client-server architectures # It will be possible to monitor tool connection server performance
Requirements	WP5_TI_004, WP5_TI_005, WP5_TI_006, WP5_TI_007 WP5_TI_008, WP5_TI_009, WP5_TI_010, WP5_TI_014, WP5_TI_015, WP5_TI_016

3.1.4.1.2 Tool Characterisation Use Cases

No specific use cases have been specified for tool characterisation in the AMASS Tool Platform. They are a specialisation of Compliance Management, where the element for which compliance needs to be managed is a tool (for CPS analysis, development, V&V, etc.) and the standard to comply with corresponds to tool qualification regulation(s).

The supported requirements are:

- WP5_TQ_001 (The AMASS Tool Platform shall allow an assurance manager to specify the needs regarding qualification for the engineering tools used in a CPS' lifecycle).
- WP5_TQ_002 (The AMASS Tool Platform shall manage evidence of tool quality).
- WP5_TQ_003 (The AMASS Tool Platform shall be to import tool quality information such as tool qualification dossiers).
- WP5_TQ_004 (The AMASS Tool Platform should indicate the tool quality needs that need to be fulfilled in a given assurance project).
- WP5_TQ_005 (The AMASS Tool Platform should indicate the degree to which tool quality requirements for the engineering tools used in a CPS' lifecycle have been fulfilled).

3.1.5 Cross-Intra Domain Reuse Use Cases

3.1.5.1 Compliance Management Use Cases

The uses cases for the Compliance Management model are responsible for the modelling, management and monitor of standards. This is done in the context of an assurance project. The tool should provide the Assurance Manager with mechanisms able model all the information contained in a standard, which includes the life-cycle defined on it, requirements and recommendations. Additionally, this building block manages the assurance project, which implies the modelling of the baseline and the compliance and equivalence mappings.

The generated compliance report allows monitoring the status of the assurance project with respect to a certain standard.

Figure 17 shows the Compliance Management Use Cases.

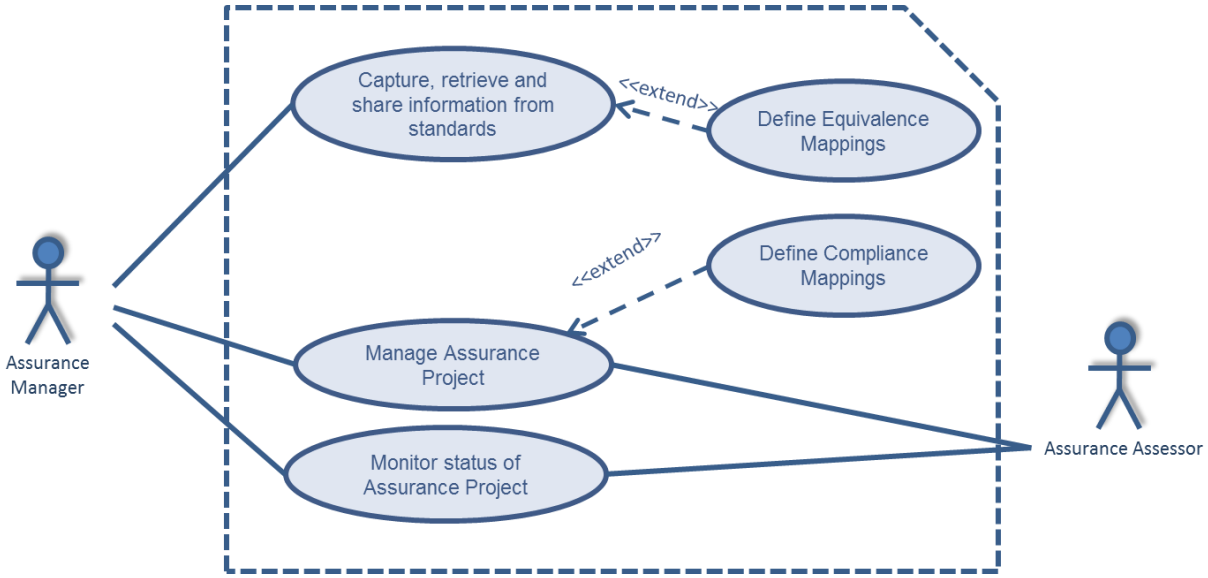


Figure 17. Use Cases for "Compliance Management" module

Table 62. Use case "Capture information from standards"

Use Case	Capture information from standards
Functionality Description	The system should be able to retrieve, digitalize and store a set of norms, recommendations, standards, or quality models.
Actors	Assurance Manager
Assumptions	A metamodel shall allow the structuration of standard information. The actor has deep knowledge about the standards. The standards information shall be always available in the platform (except if it is explicitly dropped).
Post-conditions	None
Steps	1. The user creates a new standard model and specifies the characteristics that define the standard 2. The user structures/categorizes the standard by parts, objectives, activities, practices, goals and requirements 3. The user describes the parts, objectives, activities, practices, goals and requirements contained in the standard.
Variations	# The user extends the standard interpretation by defining a project baseline. # From the baseline, the system is able to generate argument fragments for the assurance case in relation with process-based argumentation. # Import and modify an existing standard model
Non-functional	None
Requirements	WP6_CM_001, WP6_CM_002

Table 63. Use case "Manage Assurance Project"

Use Case	Manage Assurance Project
Functionality Description	The system should be able to create, modify and drop assurance information from a specific project though the project lifecycle.
Actors	Assurance Manager, Assurance Assessor
Assumptions	A model containing information from the standard shall be available in the platform.
Post-conditions	The user shall continue with the specification with the rest of the modules.

	The user is able to assign profiles to the different users of the assurance project.
Steps	1. The user creates a new assurance project 2. The user specifies the baseline in association with a standard which will be followed in the project 3. The user specifies the compliance maps/links though the project lifecycle.
Variations	# The user imports previous assurance project information.
Non-functional	None
Requirements	WP6_CM_002

Table 64. Use case "Monitor Assurance Project Status"

Use Case	**Monitor Assurance Project Status**
Functionality Description	The system should be able to provide information about the assurance activities progress in relation with the plan.
Actors	Assurance Manager, Assurance Assessor
Assumptions	An assurance project and a project baseline have been specified.
Post-conditions	None
Steps	1. The user selects the assurance project 2. From a menu, the user asks for the project progress report 3. A progress report is automatically generated from the different modules information.
Variations	# The user can ask for an impact analysis.
Non-functional	None
Requirements	WP6_CM_005

Table 65. Use case "Define Equivalence Mappings"

Use Case	**Define Equivalence Mappings**
Functionality Description	Mapping of model elements from different Ref. Frameworks. A Ref. Framework model can represent either a Standards/Rules/Regulations model or a Company-Specific Process Definition model. Mapping can occur at these levels: 1. Between a process definition and a standard, either while it is being developed, of afterwards, to analyse or demonstrate compliance. 2. Between two processes or two standards, to analyse equivalence or to derive a new one from an existing one.
Actors	Standards Expert
Assumptions	The main experts from a company, likely from supplier companies, and from external assessors must define a procedure to create interpretations of mappings
Post-conditions	None
Steps	1. Browse the Source and Target Ref. Framework models 2. Create map links between model elements of the source and target Ref. Frameworks. 3. Edit the map information, including coverage, conditions, justifications, etc.
Variations	None
Non-functional	None
Requirements	WP6_CM_008

Table 66. Use case "Define Compliance Mappings"

Use Case	**Define Compliance Mappings**

Functionality Description	The compliance mapping is the mechanism the system has to indicate that an asset (an activity of the process, a requirement, an analysis…) has been executed as mean for compliance with part of what it is requested in a standard or regulation.
Actors	Assurance Manager
Assumptions	None
Post-conditions	None
Steps	1. Browse the Source (Assurance Assets from the Assurance Project) and Target Ref. Framework models 2. Create map links between model elements of the source and target. 3. Edit the map information, including coverage, conditions, justifications, etc.
Variations	None
Non-functional	None
Requirements	WP6_CM_008

3.1.5.2 Process/Product/Assurance Case reuse via management of variability Use Cases

Figure 18 shows the Process/Product/Assurance Case reuse via management of variability Use Cases.

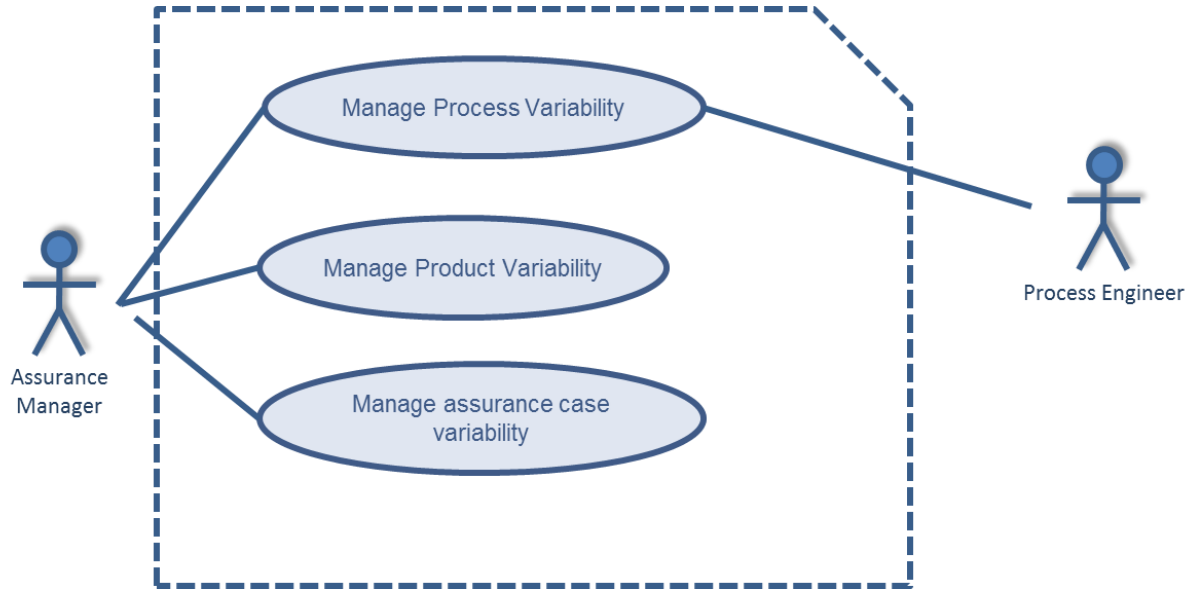


Figure 18. Use Cases for "Product/Process/Assurance Case Line Specification" module

Table 67. Use case “Manage process variability”

Use Case	Manage process variability
Functionality Description	The AMASS tools shall enable the specification/systematization of variability at the process level.
Actors	Assurance Manager, Process Engineer
Assumptions	A Process library (Base Model) has been specified
Post-conditions	None
Steps	1. The user imports the Base Model into a project. 2. The user manages variability via the variability, resolution, realization editors. 3. The user generates/export the new process model, obtained as tailoring of the Base Model.
Variations	None

Non-functional	None
Requirements	WP6_PPA_001

Table 68. Use case "Manage product variability"

Use Case	**Manage product variability**
Functionality Description	The AMASS tools shall enable the specification/systematization of variability at the product level.
Actors	Assurance Manager, Development Engineer
Assumptions	The component warehouse (Base Model) has been specified
Post-conditions	None
Steps	1. The user manages variability via the variability, resolution, realization editors 2. The user generates/export the new component model, obtained as tailoring of the Base Model
Variations	None
Non-functional	None
Requirements	WP6_PPA_004

Table 69. Use case "Manage assurance case variability"

Use Case	**Manage assurance case variability**
Functionality Description	The AMASS tools shall enable the specification/systematization of variability at the assurance case level.
Actors	Assurance Manager
Assumptions	An assurance case model (Base Model) has been specified
Post-conditions	None
Steps	1. The user manages variability via the variability, resolution, realization editors. 2. The user generates/export the new assurance case model, obtained as tailoring of the Base Model.
Variations	None
Non-functional	None
Requirements	WP6_PPA_005

3.1.5.3 Reuse Assistant Use Cases

The reuse assistance functionality concerns intra- and cross-domain reuse of assurance and certification assets. The main focus of the reuse is on the following assurance assets:

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

The concrete scenarios of reuse include:

- ***Cross-system reuse (intra-standard or intra-domain):*** reuse of assurance assets when a product or system evolves in terms of functionality or technology (e.g. product upgrade). We assume that the reusable assurance assets were compliant with the same standards we target in the new scenario.
- ***Cross-standard reuse (cross-domain or cross-concern):*** reuse of assurance assets from a project that was completed in compliance of a different dependability concern (e.g., security-compliant

assurance project reused from a safety-compliant assurance project) or different domain (e.g. avionics-compliant assurance project reused from an automotive compliant assurance project).

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

Figure 19 shows the Reuse Assistant use cases, which are documented in Table 70 to Table 73. The grey-coloured use case means that it was documented in Section 3.1.1 (cf. Table 3).

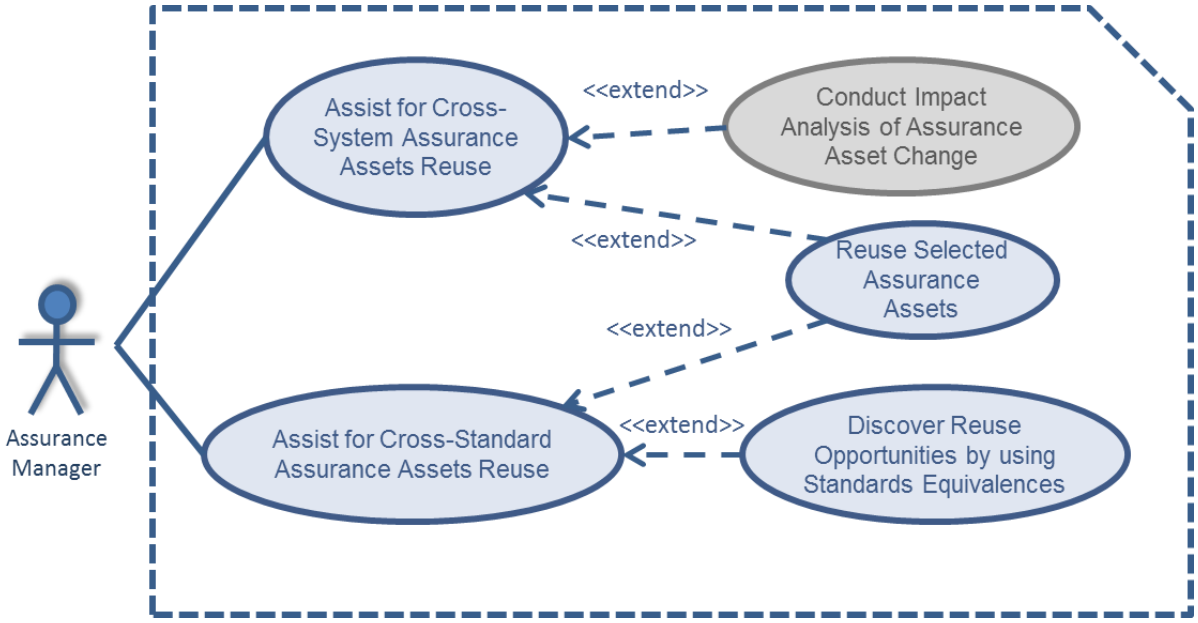


Figure 19. Use Cases for "Reuse Assistance" module

Table 70. Use case “Assist for Cross-System Assurance Assets Reuse”

Use Case	Assist for Cross-System Assurance Assets Reuse
Description	The actor is able to reuse assurance assets from one assurance project into another. This functionality relates to the cross-systems reuse scenario in which both source and target assurance projects must be compliant to the same standard or set of standards. To perform cross-system reuse, the actor can conduct impact analysis (a separate use case documented in previous sections) in order to understand the consequences of the reuse operation and to identify the set of assurance assets that may be affected when executing the reuse operation. Once the actor selects the assurance assets to be reused, the reuse operation itself can be executed.
Actors	Assurance Manager
Assumptions	• A previous assurance project (source project) has been created in the AMASS platform, which includes assurance assets (evidence, process, argumentation, compliance models) and has been eventually completed. • A new assurance project has been created (target project)
Post-conditions	Different AMASS models will be updated according to the reuse scope, including evidence models, argumentation models, process models and compliance information.
Steps	1. The actor opens the newly created assurance project (target project), starts the reuse assistant, and selects the reusable assurance project (source project). 2. A number of assurance models from the source project are available for navigation, including evidence, process, argumentation and baseline (compliance

	information) models and can be individually selected. 3. Once a specific model in the source project is selected, the actor can call the impact analysis functionality to select model elements and visualise the dependent (impacted) model elements. 4. Once selected all the desired model elements to be reused from the various models, the actor can store the subset before executing the reuse operation.
Variations	# Additional selection criteria can be applied such as e.g. subset of assurance assets associated to a given criticality level.
Non-functional	None
Requirements	WP6_RA_001, WP6_RA_005

Table 71. Use case "Assist for Cross-Standards Assurance Assets Reuse"

Use Case	**Assist for Cross-Standards Assurance Assets Reuse**
Description	The actor is able to reuse assurance assets from one assurance project into another, when they relate to different industry domains, dependability concerns or industry standards in general. To perform cross-standard reuse, an equivalence map model must be created before, between the source and the target standard models. The reuse assistant provides information on the reuse opportunities as result of the equivalence relationships. Once the actor selects the assurance assets to be reused, the reuse operation itself can be executed.
Actors	Assurance Manager
Assumptions	• A previous assurance project (source project) has been created in the AMASS platform, which includes assurance assets (evidence, process, argumentation, compliance models) and has been eventually completed. • It exists an equivalence map model between the source and target standards (see previous sections for creating equivalence maps) • A new assurance project has been created (target project). It is mandatory that the target assurance project is based in a standard model with equivalence maps with the standards model in which is based the source assurance project.
Post-conditions	Different AMASS models will be updated according to the reuse scope, including evidence models, argumentation models, process models and compliance information.
Steps	1. The actor opens the newly created assurance project (target project), starts the reuse assistant, and selects the reusable assurance project (source project). 2. Assurance models from the target project can already exist or it may create them automatically. In the latter case, the model elements will follow the same structure and naming as the standards (baseline in this case) model. 3. A number of assurance models from the target project are available for navigation, including evidence, process, argumentation and baseline (compliance information) models and can be individually selected. 4. One a source model element is selected, the actor can discover reuse opportunities by using equivalence maps (extended use case). 5. Once selected all the desired model elements to be reused from the various models, the actor can store the subset before executing the reuse operation. Storing the subset of assets permits to check the actions that need to be performed before executing them. This is important for example if it is necessary to have an internal approval or if the user wants to check the impact of the operation before executing it.
Variations	None

Non-functional	None
Requirements	WP6_RA_002, WP6_RA_003, WP6_RA_004

Table 72. Use case "Discover Reuse Opportunities by using Standards Equivalences"

Use Case	**Discover Reuse Opportunities by using Standards Equivalences**
Description	The actor can be assisted to identify reuse opportunities between assurance projects, when they are assessed against different standards. The mechanism to discover reuse opportunities is by using the equivalence map models associated to standard models.
Actors	Assurance Manager
Assumptions	There exists an equivalence map model between the source and target standards (see previous sections for creating equivalence maps).
Post-conditions	Identification of model elements with associated equivalence standard model elements.
Steps	1. Select target model elements 2. Visualise the equivalent model elements in the source assurance projects 3. Look at the reuse post-conditions identified in the equivalence map model. 4. Decide if the reusable element will be selected for reuse.
Variations	None
Non-functional	None
Requirements	WP6_RA_002, WP6_RA_003, WP6_RA_004

Table 73. Use case "Reuse Selected Assurance Assets"

Use Case	**Reuse Selected Assurance Assets**
Description	Actors can take a selected set of assurance assets (source assurance project) and perform the operation of copy into the target assurance project.
Actors	Assurance Manager
Assumptions	A subset of assurance assets has been selected
Post-conditions	Copy operation in the AMASS repository
Steps	1. Visualise the subset of selected assurance assets 2. Perform reuse operation 3. Visualise results of the reuse operation
Variations	None
Non-functional	None
Requirements	WP6_RA_001, WP6_RA_002, WP6_RA_003, WP6_RA_004, WP6_RA_005

3.2 Structural View (*)

This section describes the structure of architecturally-significant modules of the ARTA platform.

Figure 20 shows the AMASS Platform Structural View. Please note that this architecture maps to the functional areas illustrated in Figure 1 and Figure 2. The only new tool module is "AMASS HMI", which is the Presentation layer (GUI) of the AMASS platform. Then, "Application" level has been identified which consists of the core AMASS building blocks. The Infrastructure layer provides services such as access and security, reporting, and system administration. Furthermore, project level management and traceability are pursued.

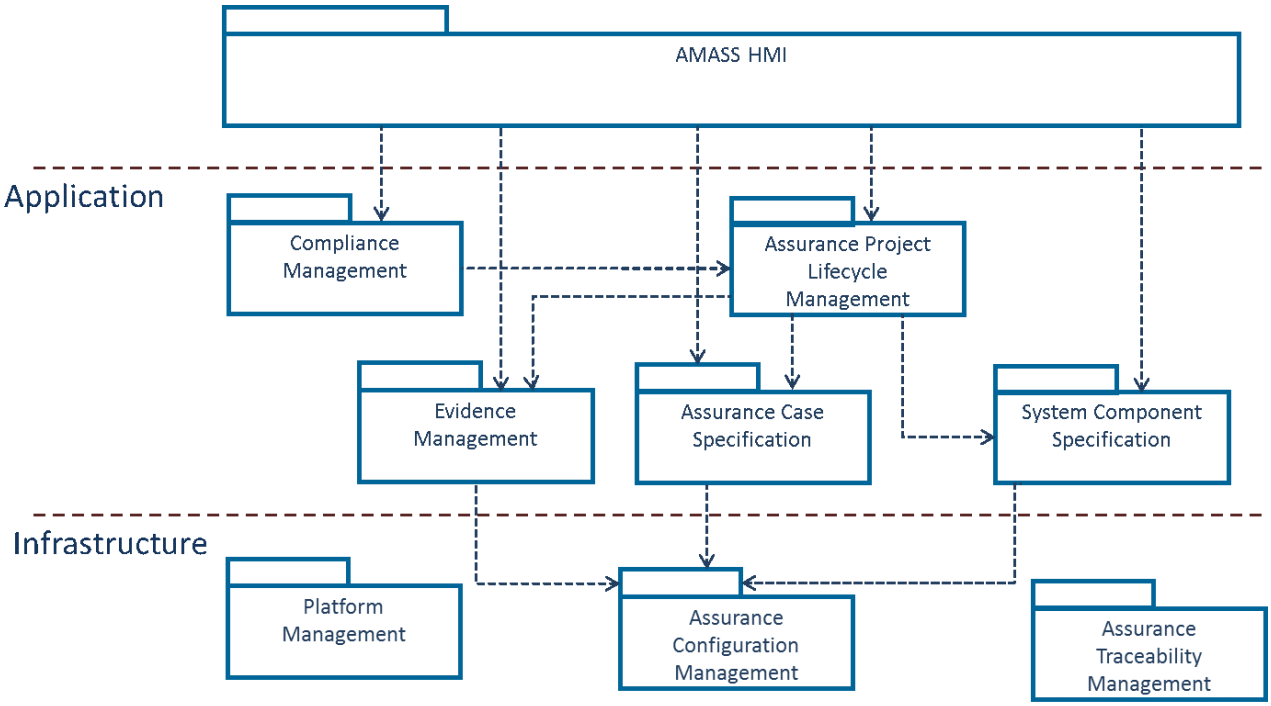


Figure 20. AMASS Structural View

3.2.1 Infrastructure Module

3.2.1.1 Assurance Project Lifecycle Management Component

There is only one tool component in this module: the *Project Framework Manager*. It includes services to capture the plans for compliance including general information, project roles, managed artefacts, and assurance cases, among other aspects.

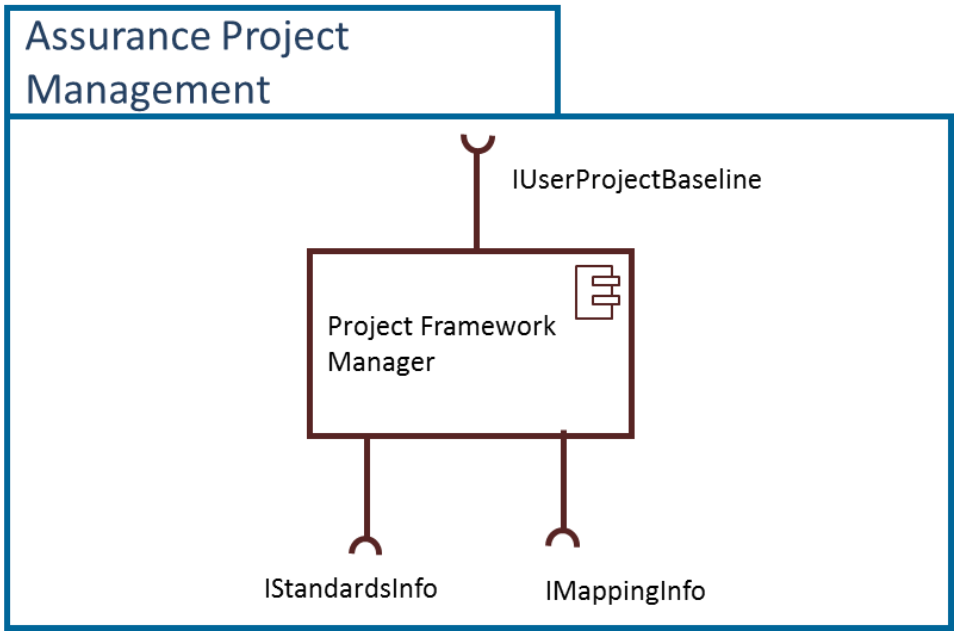


Figure 21. Tool components for Assurance Project Lifecycle Management

3.2.1.2 Assurance Traceability Components

There are two tool components that are responsible for the functionalities provided by the Assurance Traceability Module. It includes features for Assurance Assets traceability, which is highly connected with the impact analysis feature. In addition to specification and analysis services, these tools will also provide specific services for visualisation of the information in different formats.

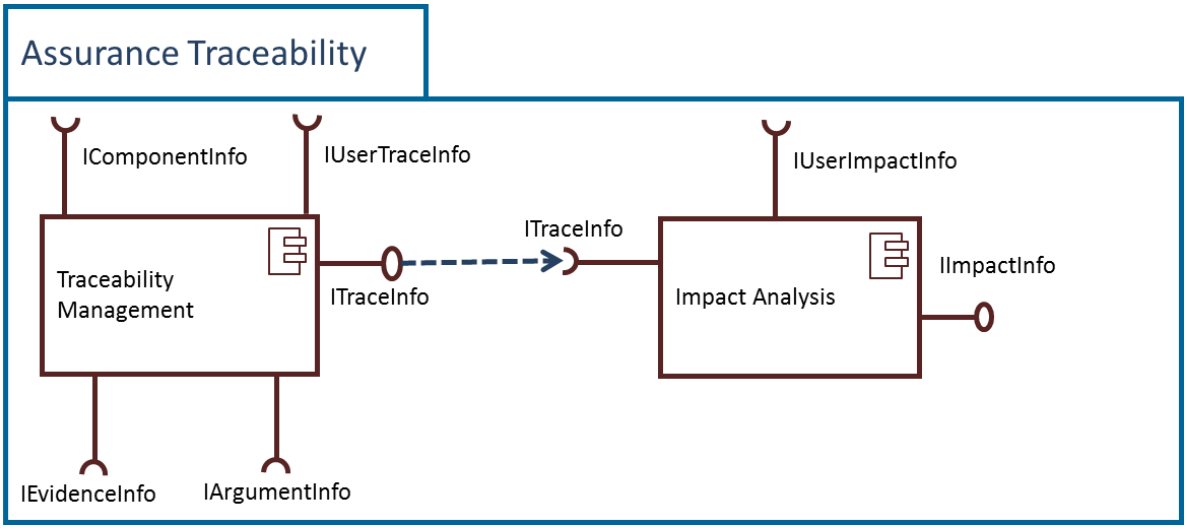


Figure 22. Tool components for Assurance Traceability

3.2.1.3 Platform Management Components

The platform management component consists of four tool components. The *Collaborative work tool* component allows different users to work at the same time with the same pieces of data. The *Data management tool* component provides features to edit, store and retrieve the data and information inside the platform, ensuring consistency and integrity. The *Reporting tool* component provides features to generate reports based on the information stored in the platform. Finally, the *Access management tool* component takes care of the log in features and of ensuring a secure access to the data.

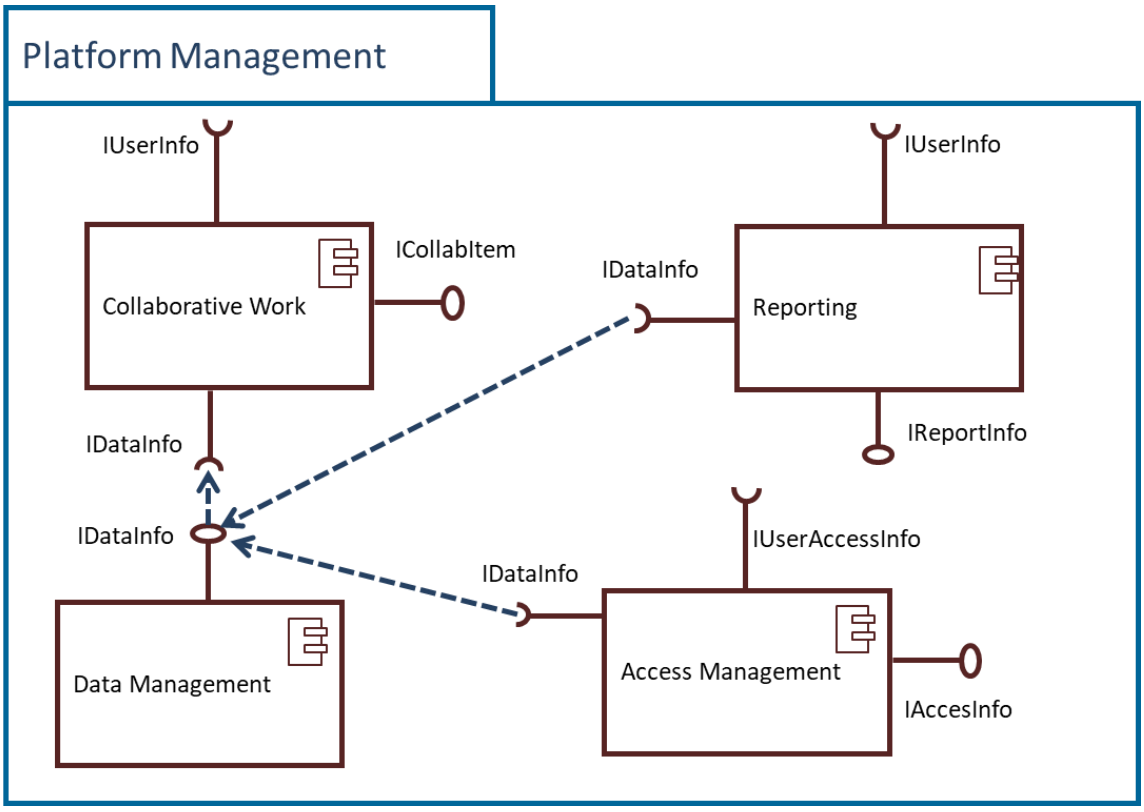


Figure 23. Tool components for Platform Management

3.2.2 Application Module

3.2.2.1 System Component Specification

We have decomposed this module into two tool components: *Component Editor* and *Product-based Arguments Generator*. The first tool component includes services to compute information about the component specification in relation to assurance needs. It is considered as an extension to common system specification tools, but with the particularity that it captures and manages information about component compliance and assurance, and provides information for the assurance case. The second tool component, the Product-based arguments generator is responsible to generate some argument fragments based on the component information specified in the first tool component.

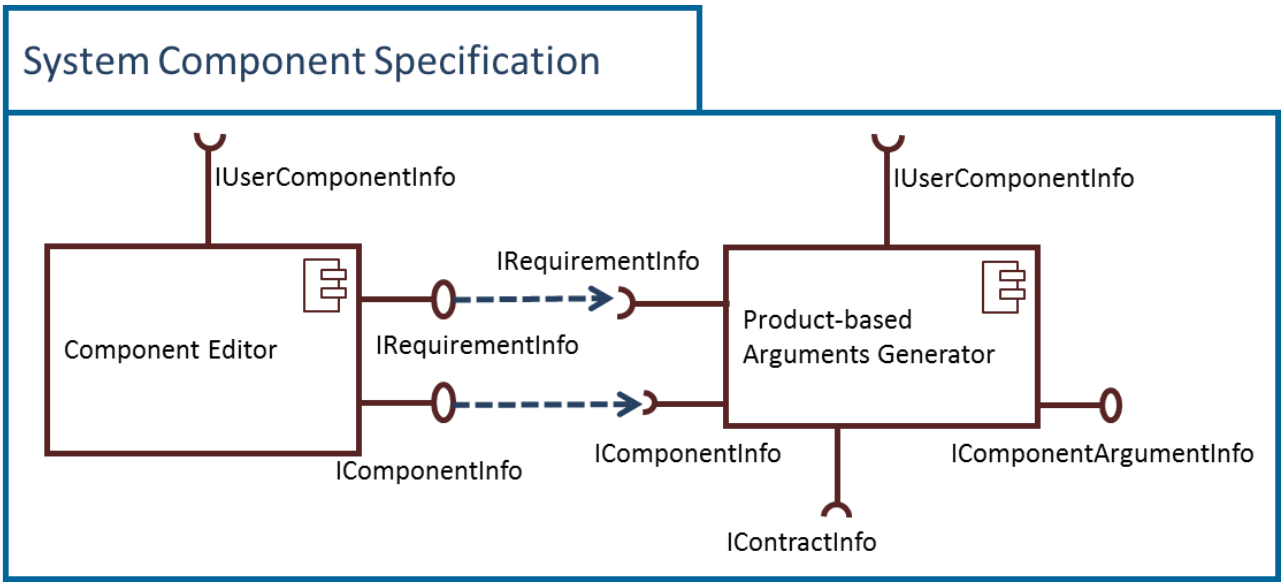


Figure 24. Tool components for System Component Specification

3.2.2.2 Assurance Case Specification Components

We have decomposed this module into three tool components: *Argument Pattern/Modules Management, Dependability modelling* and *Assurance Case Editor*. The *Argument Pattern/Modules Management* tool component is responsible for storing, creating, editing and instantiating both argument patterns and argument modules. The *Assurance Case Editor* tool component provides the user the ability to for specifying arguments using GSN graphical notation. *Dependability modelling* module supports the capability to show the dependency and impact of some decisions in the arguments and includes annotations related to multiple concerns treated in the arguments in order to make certain argumentations or design decisions better understandable. Details of the functionality and design are included in D4.3 [15], while the information about the implementation will be included in D4.6 Prototype for multiconcern assurance (c).

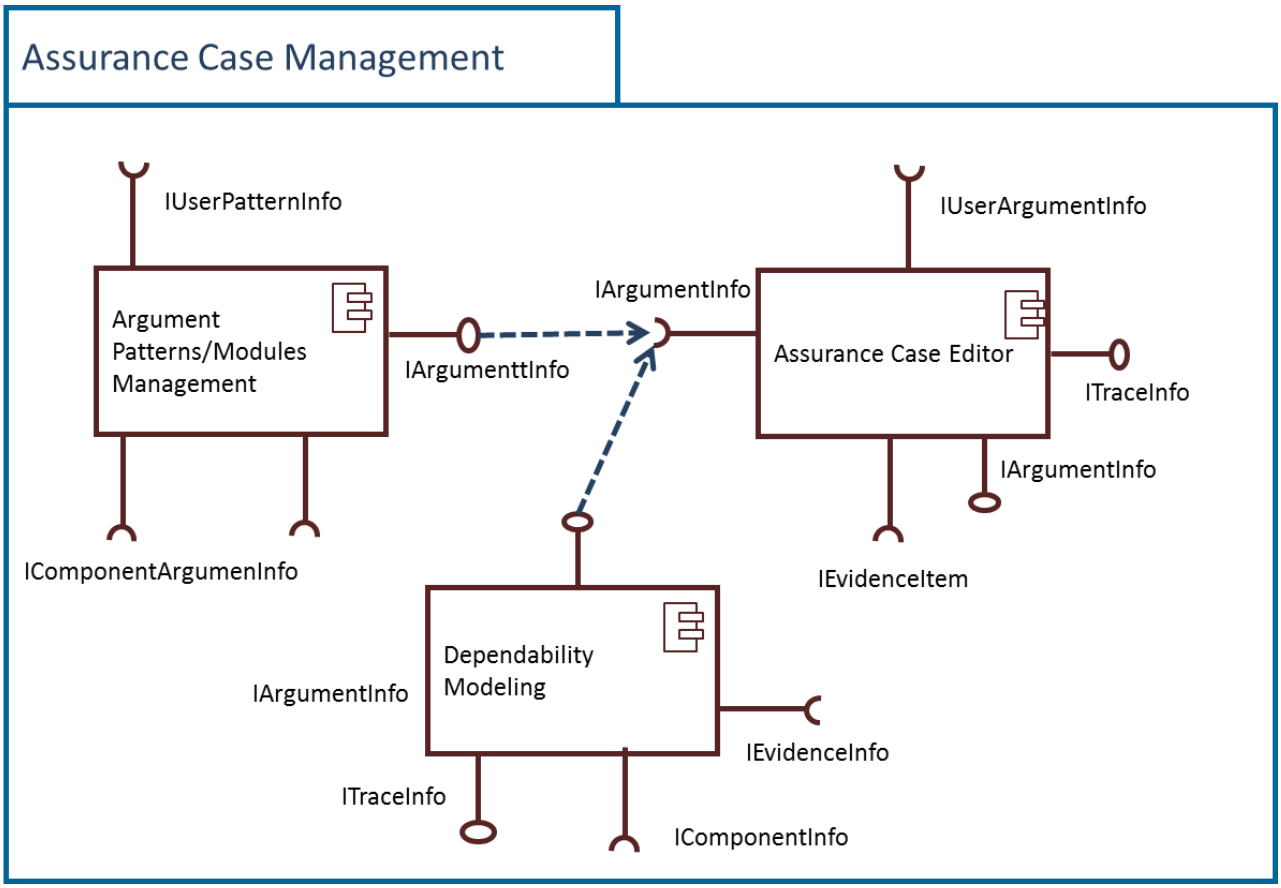


Figure 25. Tool components for Assurance Case Specification

3.2.2.3 Compliance Management Components

We have decomposed this module into three components: *Standards Editor*, *Process Editor* and *Compliance Editor*. The first tool component includes services to capture the knowledge from the standards, while the second captures information of the life-cycles that are described in the process. The focus of the third component, the *Compliance Editor*, is to use the basic information from the standards to capture the links and a mapping between which has been planned and which has been executed, in the context of a given assurance project.

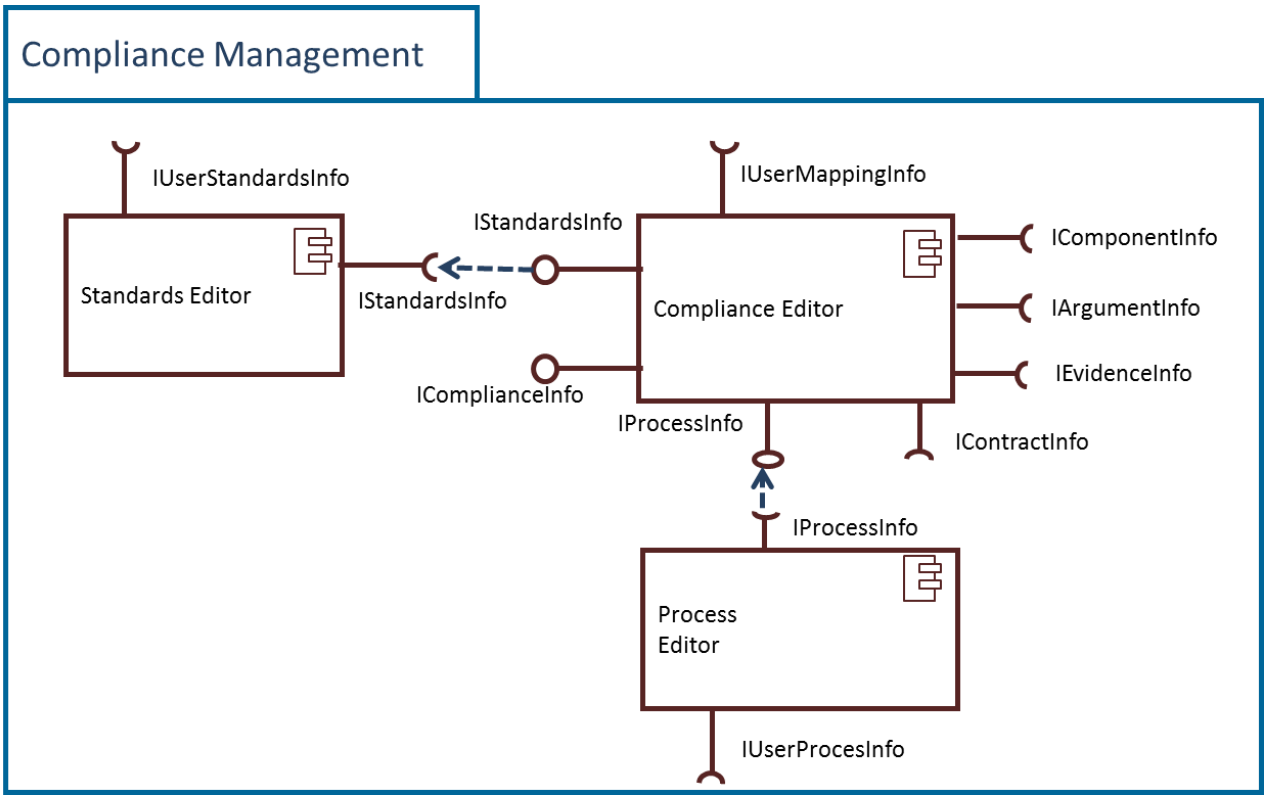


Figure 26. Tool components for Compliance Management

3.2.2.4 Evidence Management Components

This module consists of a single component: *Evidence Characterization Editor*. It provides all the necessary services for evidence storage in the AMASS Tool Platform (determination, specification, evaluation, etc.). The managed information is further used by other components, as evidence information might be necessary for e.g. support the claims of an assurance case.

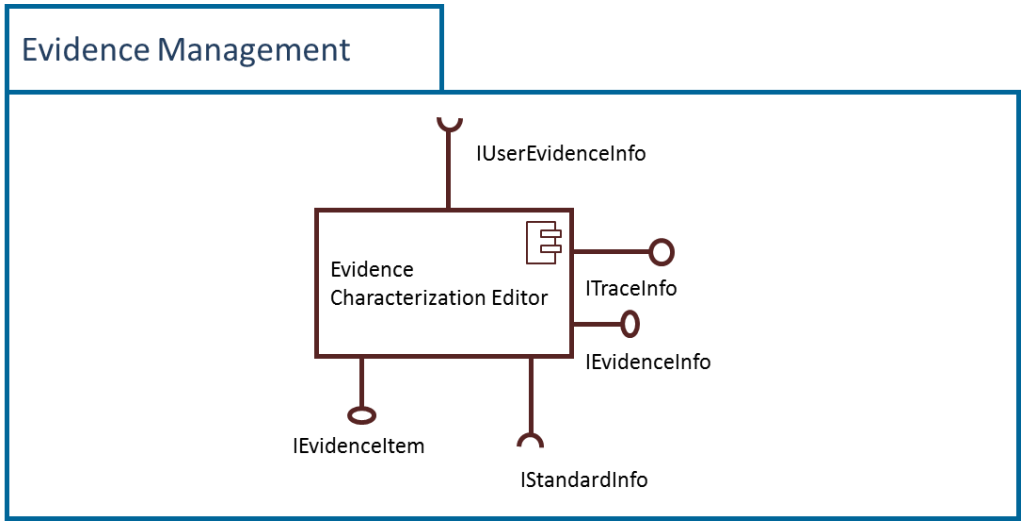


Figure 27. Tool components for Evidence Management

3.2.2.5 Seamless Interoperability Components

Two components provide the services necessary for tool integration. *Tool chain Management* supports the needs arising from having to specify and later use a set of tools together for a specific CPS engineering or assurance activity. This can include one or several system lifecycle phases. *Tool Connector* generalises all the specific connectors that will be required with specific external tools (e.g. Simulink) or via specific technologies (e.g. OSLC). These connectors will typically consume services from the external tools, but also provide some additional AMASS-specific services (e.g. for artefact search).

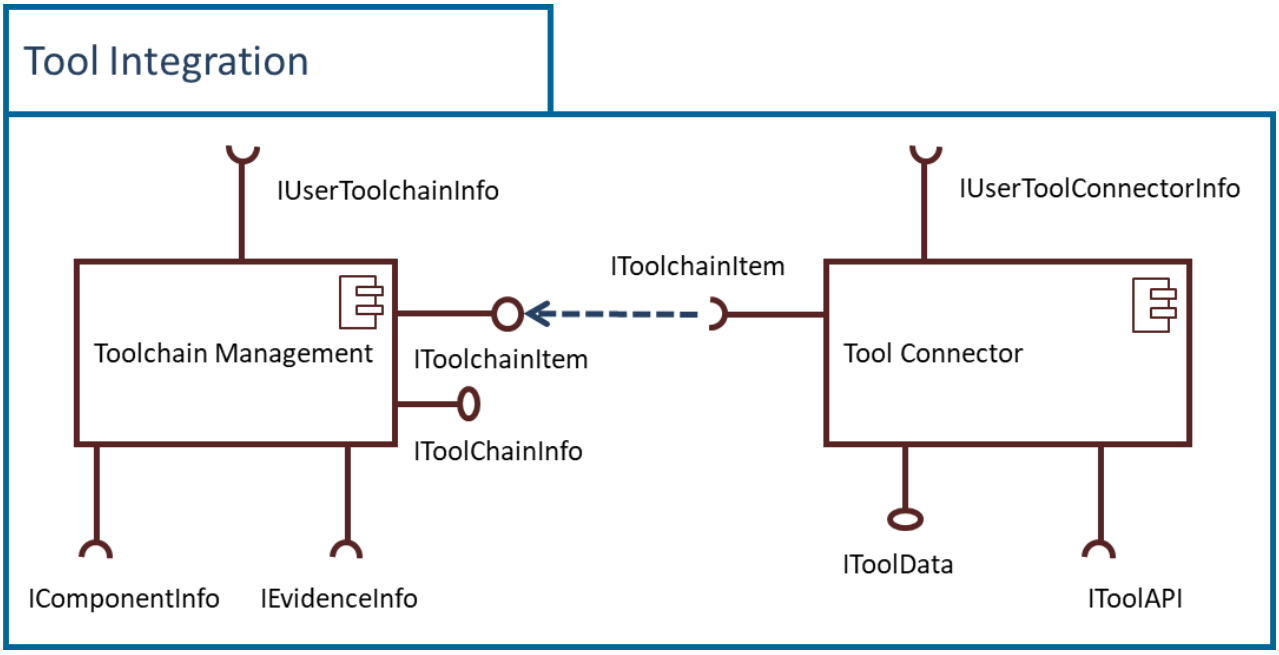


Figure 28. Tool components for Tool Integration

3.2.2.6 Contracts Management Components

This module encapsulates all the functionality related to contract management. We have two different tools related to contracts, one about the components composition, *Contract Editor*, and the other one about arguments contracts. The first tool component, the *Contract Editor* uses the information about the component requirements to capture component contracts. The *Contract-based multi-concern assurance* component tool is responsible of classifying the different contracts into the different concerns and generate the appropriate argumentation related to those contracts. Tagging contracts depending on the properties included, would let the user argue about the rationale behind possible interferences between different concerns. Tagging contracts depending on the properties included, would let the user argue about the rationale behind possible interferences between different concerns.

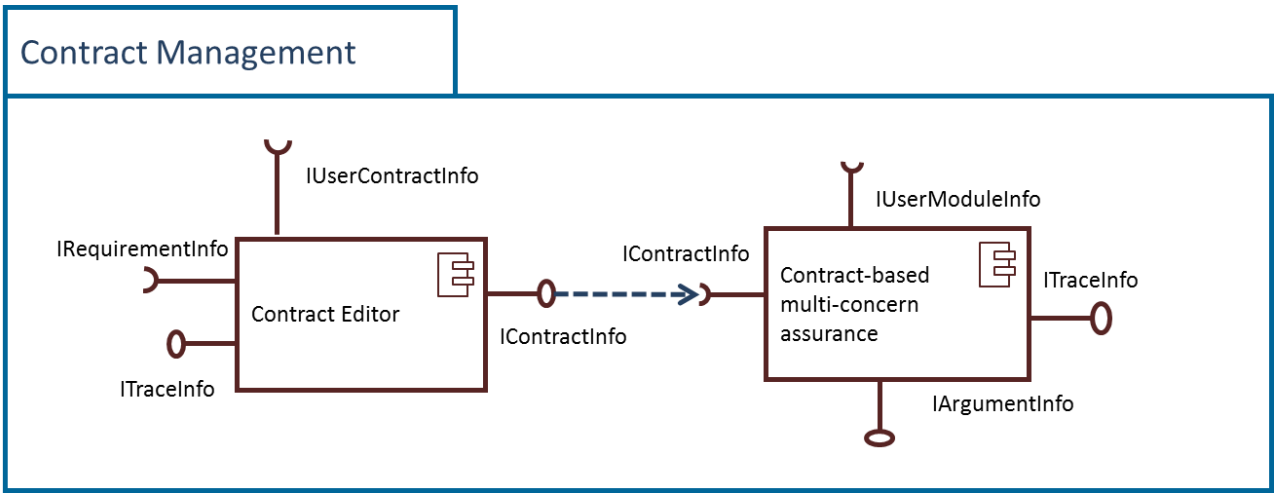


Figure 29. Tool components for Contracts Management

3.2.2.7 Assurance Analysis/Assessment

This is one of the AMASS-specific building blocks. It partially covers the challenges regarding STO1 Architecture driven assurance and STO2: Multi-concern Assurance. Details of the functionality and design are included in D3.3 [14] and D4.3 [15] while information about the implementation will be included in D3.6 and D4.6. System *Dependability Co-Analysis/Assessment* takes care of specification of different concern analysis, connect with external tools to execute the analysis and retrieve the results to be provided to the rest of the AMASS blocks. The *V&V Analysis* tool component provides services to set-up and invokes V&V analysis through the ARTA (IV&VAnalysis provided interface). In particular, it is responsible for retrieving information from the component editor (IComponentInfo required interface), mentioned in the System Architecture Management Components, and providing it to specific V&V analysis tools, in some cases external tools (IToolConnector required interface), in order to generate/report validation and verification results (IAnalysisResutls provided interface). Moreover V&V stores obtained results as evidence items (IEvidenceItem required interface) and creates traceability information between pieces of evidence produced by the analysis and the system level entities used to perform the analysis (ITraceInfo required interface).

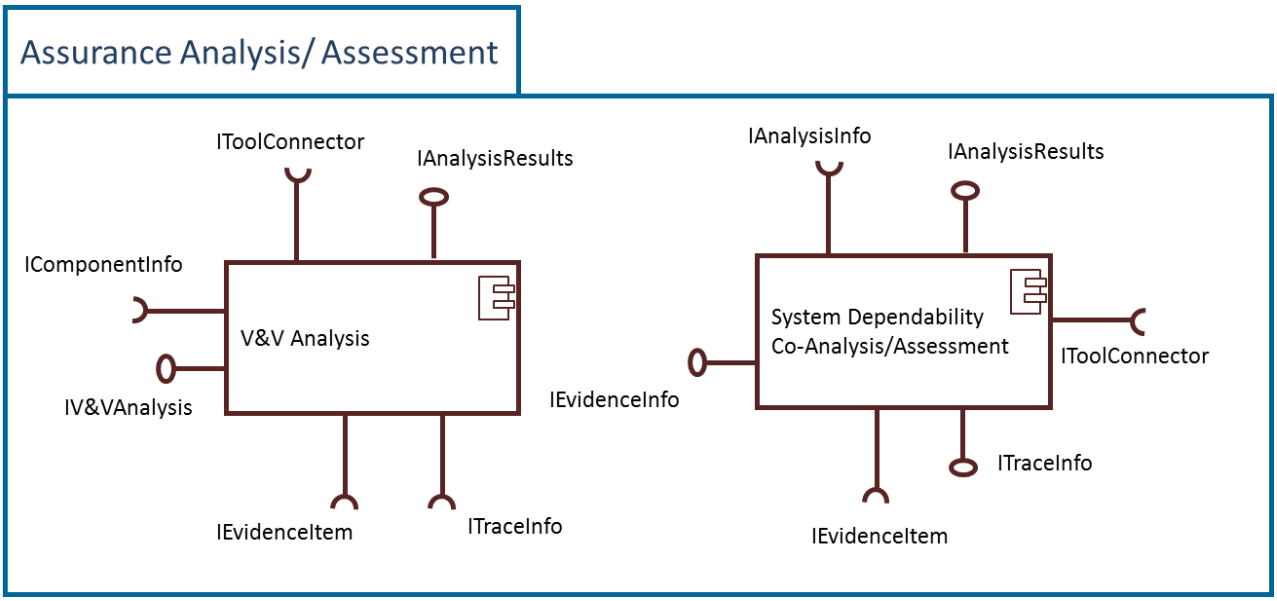


Figure 30. Tool components for Assurance Analysis/Assessment

3.2.2.8 Cross-Intra Domain Reuse Components

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

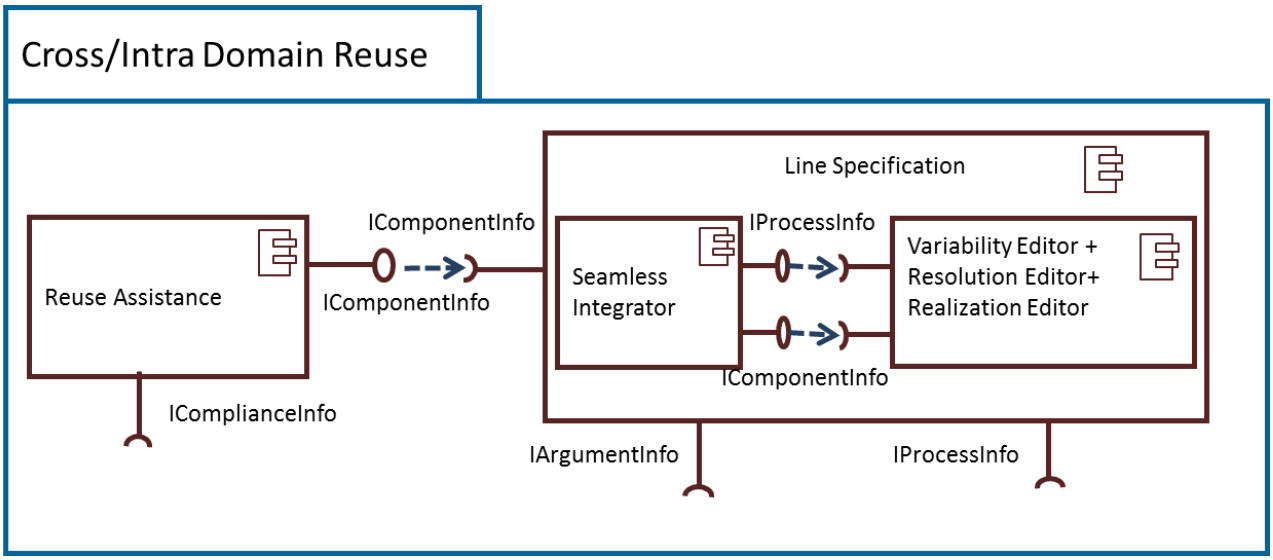


Figure 31. Tool components for Cross/Intra domain reuse

3.3 Interactional View

3.3.1 Interaction Assurance Case Development Scenario

The AMASS Platform user starts the process by creating a new Assurance Case. In the *Assurance Case Editor*, the user will create an assurance case user in the assurance case diagram by following the use case "Define and navigate an assurance case structure". Internally the assurance case editor will call the *Argument Patterns/Modules Management* tool and provide the result back in the *Assurance Case Editor*. Then the user could "Apply an argument pattern", calling again the *Argument Patterns/Modules Management* tool, and the results again will be shown in the Assurance Case Editor. The user can then "Develop claims and links to evidence"; in this phase, one of the steps will ask the user to map the evidence included in the diagram with actual pieces of evidence. So, the *Assurance Case Editor* will call the *Evidence Editor* to open the evidence model associated with the project and let the user select a piece of evidence, or view and/or update a previous selected one.

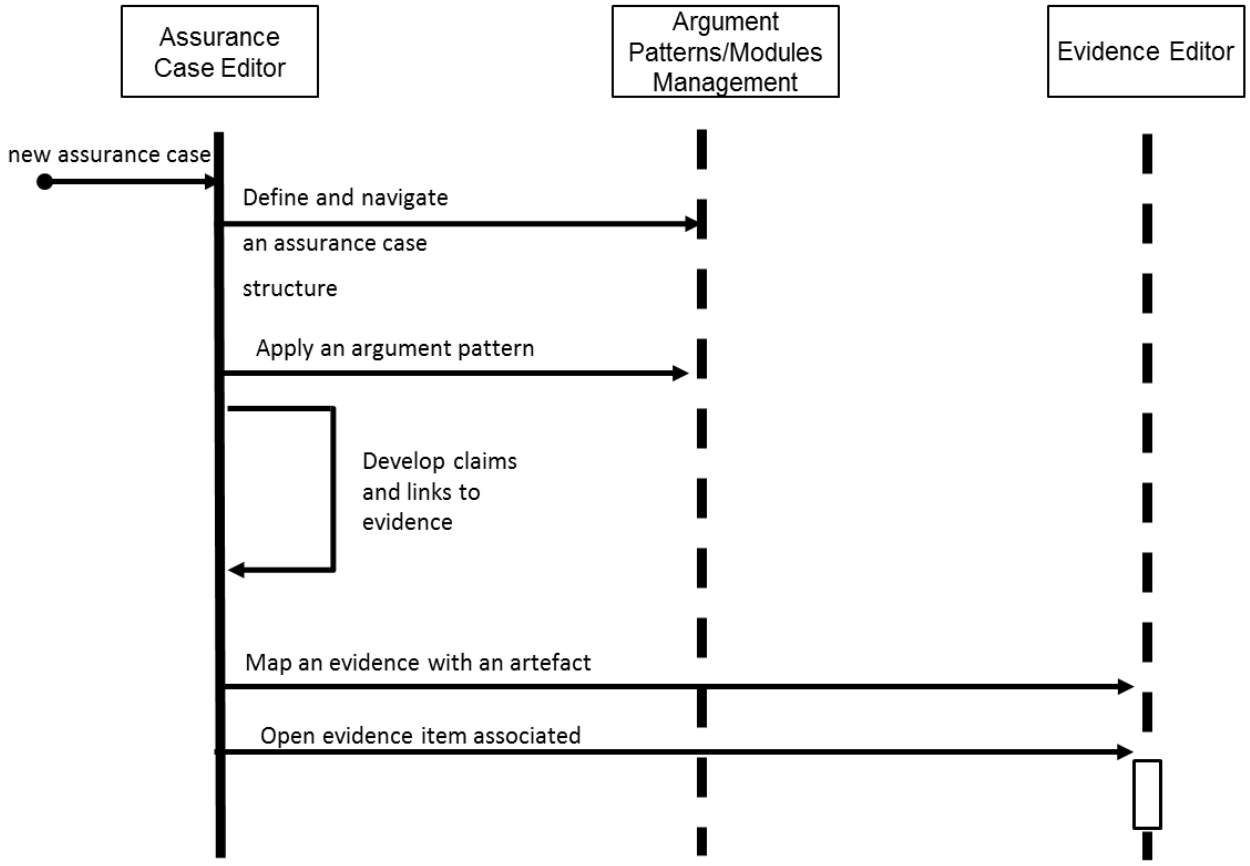


Figure 32. Interactions between tool components in the assurance case development scenario

3.3.2 Interaction Modular Argumentation/Architecture Development Scenario

The AMASS Platform user starts the process by creating a new Assurance Case. In the *Assurance Case Editor*, the user will create an assurance case in the assurance case diagram by following the use case "Define and navigate an assurance case structure". Internally the assurance case editor will call the *Argument Patterns/Modules Management* tool and provide the result back in the *Assurance Case Editor*. Then the user could "Reuse an argument module", calling again the *Argument Patterns/Modules Management* tool, and the results again will be shown in the Assurance Case Editor. The user can then "Associate assurance case with component specification"; in this phase the user maps each the assurance case with a specific component design included in the *System Component tool*. The user can go on "Editing the component specification" and at any time, and can request to "*Show associated component assurance case*", which will take the user back to the *Assurance Case Editor* to browse the assurance case diagram.

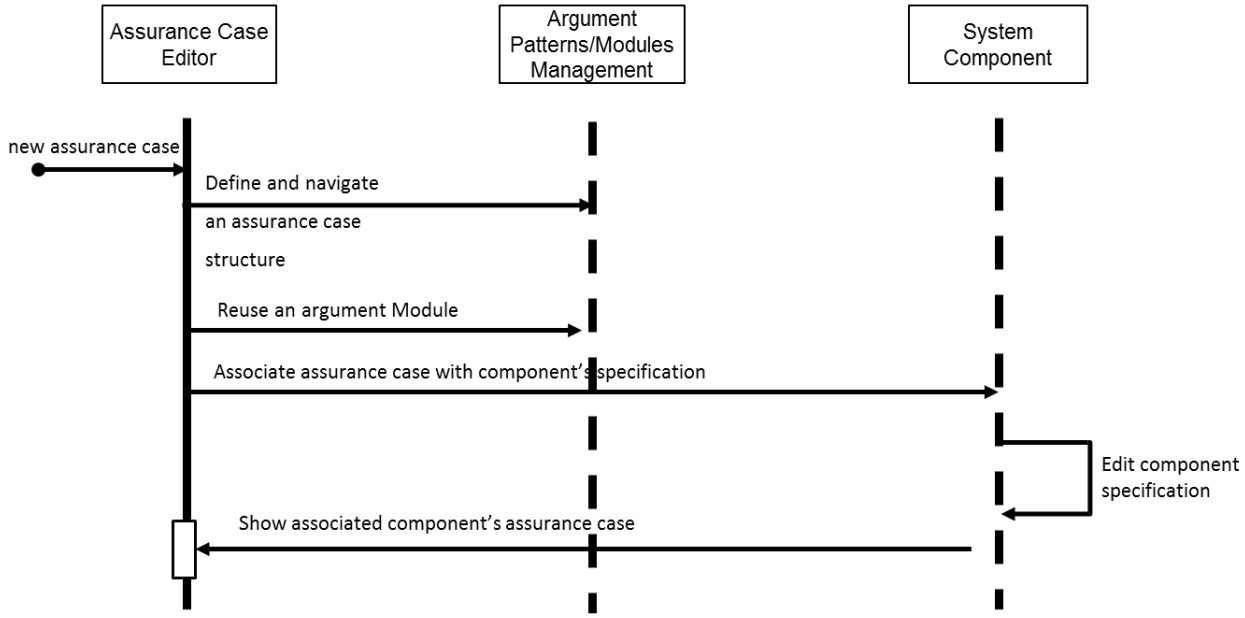


Figure 33. Interactions between tool components in the development modular argumentation/architecture scenario

3.3.3 Interaction Process Based Argumentation Scenario Development

The user starts this scenario by creating a new process model in the *Process Editor*. This model will specify the process that will be followed in the project. From the *Process Editor*, the user then can request the "Automatic generation of process arguments". This will open the *Assurance Case Editor* and show the diagram with the new argument fragment generated included in the assurance case diagram associated with the project.

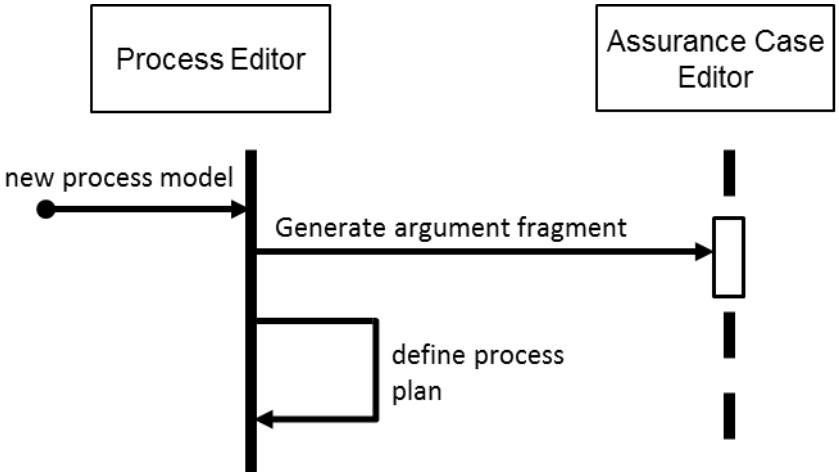


Figure 34. Interactions between tool components in the development process based argumentation scenario

3.3.4 Interaction Multi-Concern Co-Analysis/Assessment Scenario

The AMASS platform user starts this scenario by editing an assurance case diagram in the *Assurance Case Editor*. In the *Assurance Case Editor*, the user can go on "Defining claims and evidence". The user from the *Assurance Case Editor* can request to "Define a safety/security analysis". The platform will open the *Co-Analysis/Assessment* tool where the user can "Perform the analysis". The *Co-Analysis/Assessment* tool will "Provide evidence results" to both the *Assurance Case Editor* which will associate it to the specific evidence in the diagram and to the *Evidence Editor* to be included in the artefacts model.

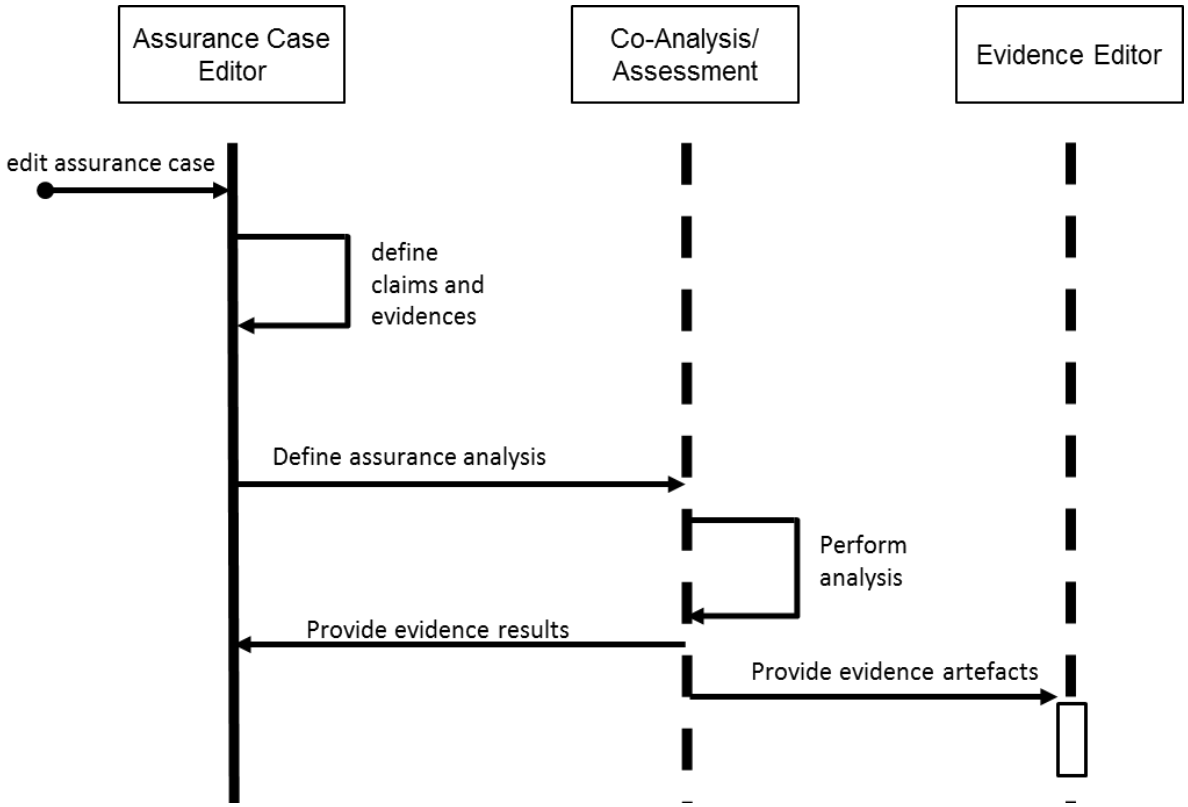


Figure 35. Interactions between tool components in the Multi-concern co-analysis/assessment scenario

3.3.5 Interaction Development Cross/Intra-Domain Reuse of Base Models

In order to perform the cross/intra-domain reuse, via the Line Specification component, the base model is first cleaned via the Seamless Integrator. As extensively explained in D6.2 [13], the base model could represent a process, a component-based specification, or an assurance case. Once cleaned, such model can be manipulated by the set of editors responsible for variability management. In particular, the variability in the base model is used by the realization editor to specify the fragment substitutions. A consistent exchange between realization and resolution editors takes place; therefore, the execution of fragment substitutions is based on the inclusion/exclusion of associated choices in the configuration/resolution. The consistent exchange between resolution and variability editor takes place to check whether the resolution satisfies the constraints, expressed in the model edited via the Variability Editor. Note that the specific configuration/resolution is executed to generate the tailored model. The tailored model is exported back to the base model editor. Further details regarding the interplay of the set of editors can be found in D6.2 [13].

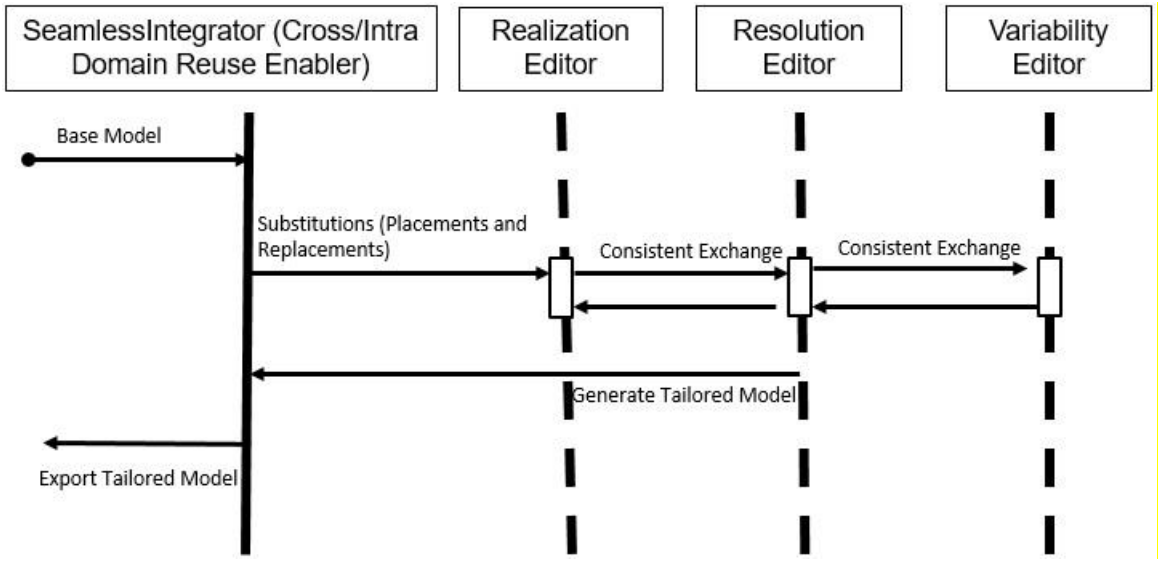


Figure 36. Interactions between tool components in the development Cross/Intra Domain Reuse of Base Model scenario

3.3.6 Interaction User Access and Concurrent Evidence Management

This interaction presents a possible series of steps for the scenario in which a user is managing evidence in the AMASS Tool Platform and, at the same time, another user is accessing the same data. Collaborative work could be performed on other assurance assets, e.g. assurance cases. The user needs to first log in the Platform. Next, the collaboration can start, and the concurrent modifications will be notified.

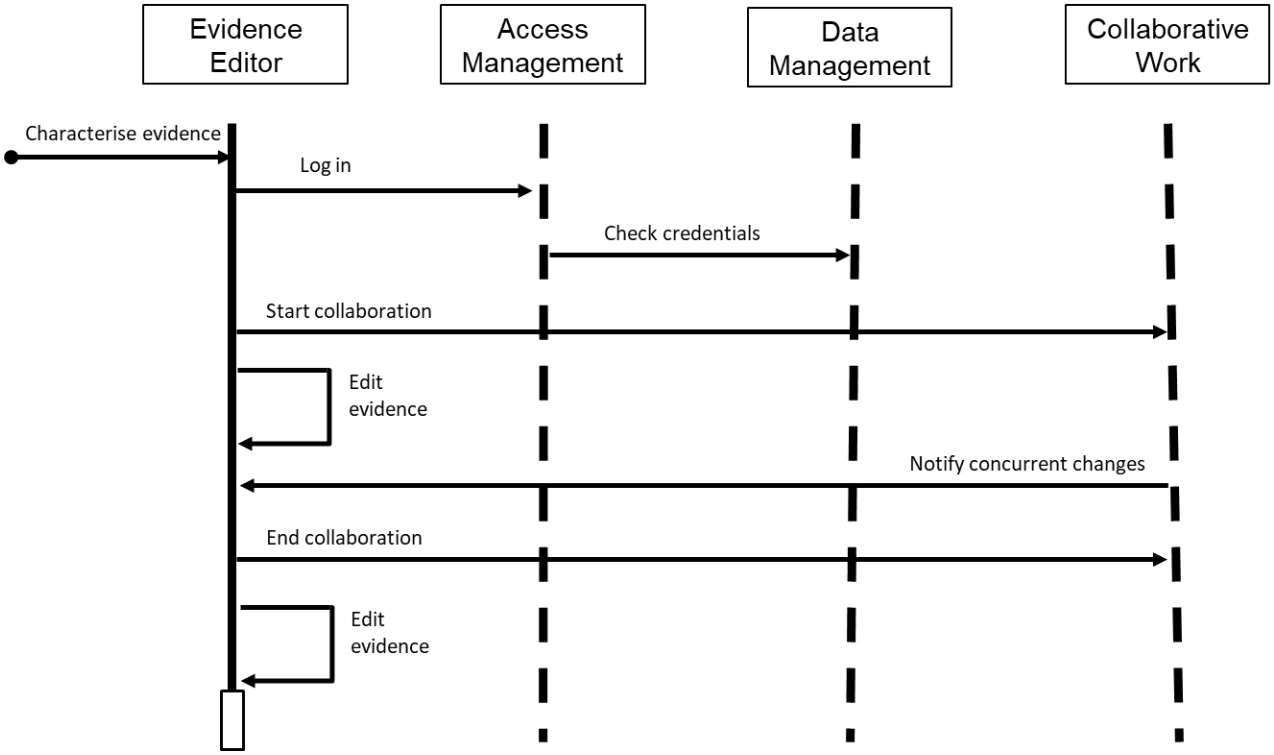


Figure 37. Interactions for user access and concurrent evidence management

3.3.7 Interaction Evidence Information Exchange and Traceability

This interaction corresponds to the situation in which a user aims to exchange evidence information with an external tool and perform traceability activities. It is an example of data exchange and of traceability support possibilities.

The user needs to determine the tools that will be part of the enacted tool chain. Tool connectors are also necessary. Afterwards the necessary data can be imported and used to characterise evidence. As part of evidence management, trace links between evidence artefacts might be specified, and change impact analysis would be conducted as result of evidence artefact modifications. All the resulting changes might have to be registered in external tools too, i.e. the data in the external tools might have to be updated.

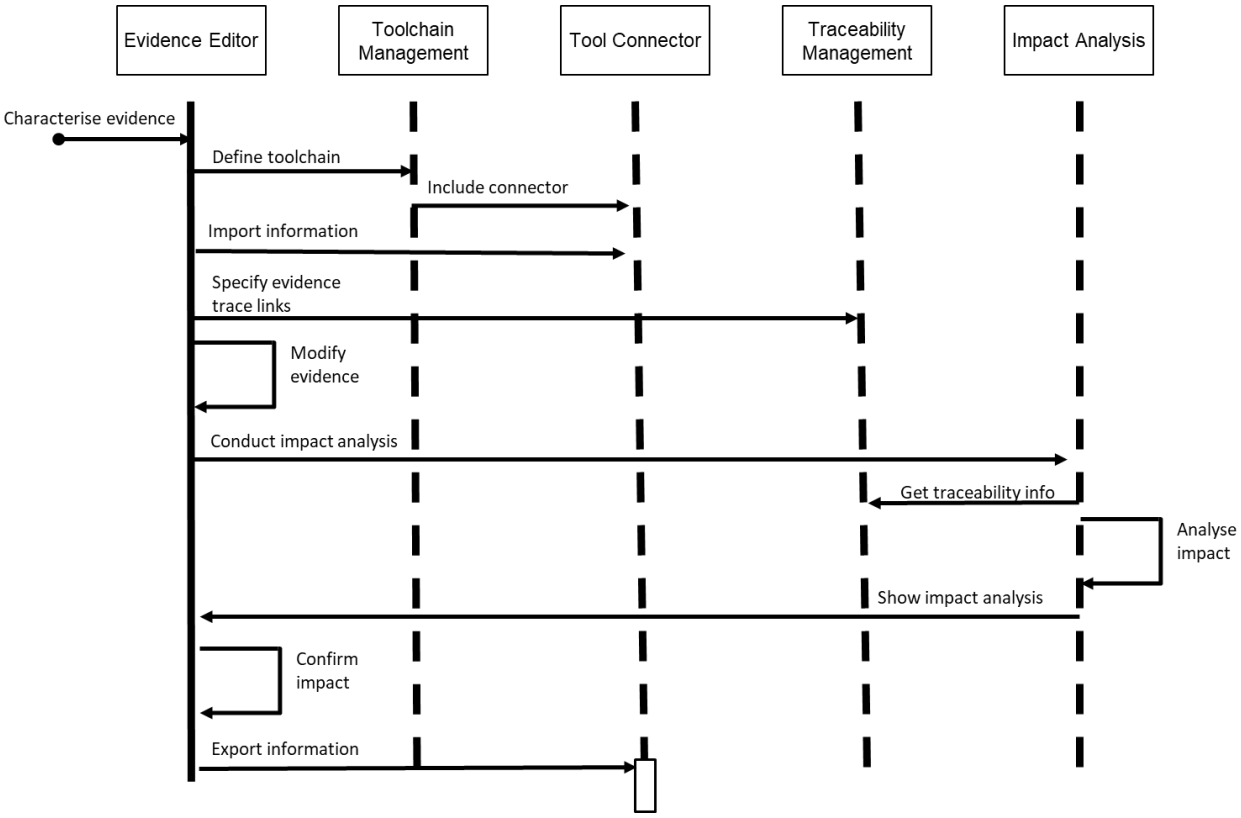


Figure 38. Interactions for evidence information exchange and traceability

3.3.8 Interaction Invoke V&V Analysis

This scenario details the collaboration envisaged for the realization of the use cases described in section 3.1.2.6, in particular regarding the request of execution of a given V&V analysis.

When the aforementioned request is sent to the V&V Analysis component, the latter retrieves the proper information regarding the architecture to be checked, and then invokes the tool connector in charge to execute the analysis (for instance by first adapting the information set and then invoking the dedicated external tool). The analysis results obtained from the tool connector invocation are then propagated from ToolConnector to V&VAnalysis and properly showed. In case the results are satisfactory (e.g. this can be decided by the ARTA user), the proper evidence item and traceability information (e.g. between the evidence itself, the analysis tool and the architecture related information) are created.

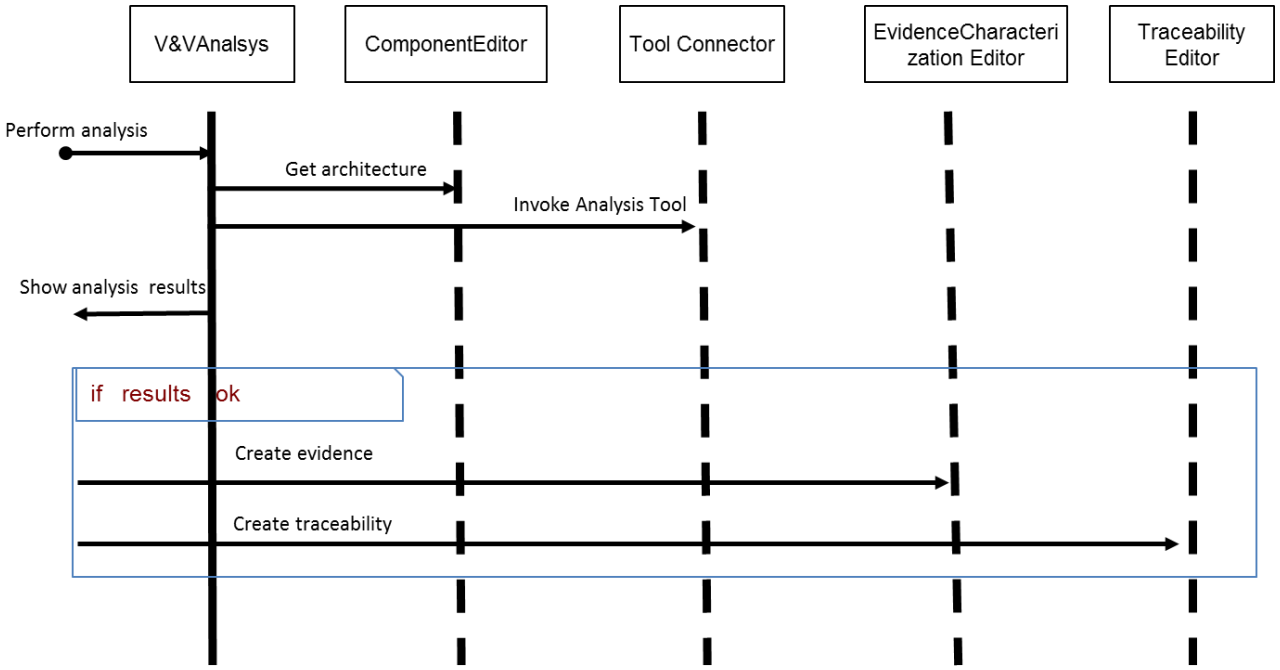


Figure 39. Interactions for invoke V&V Analysis

4. Conceptual CACM (*)

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 40. Note that there is not a 1:1 relationship between the views introduced here and the metamodels presented in the following subsections, although the overall intent of the models accords with that in Figure 40. The metamodels are presented in sections below.

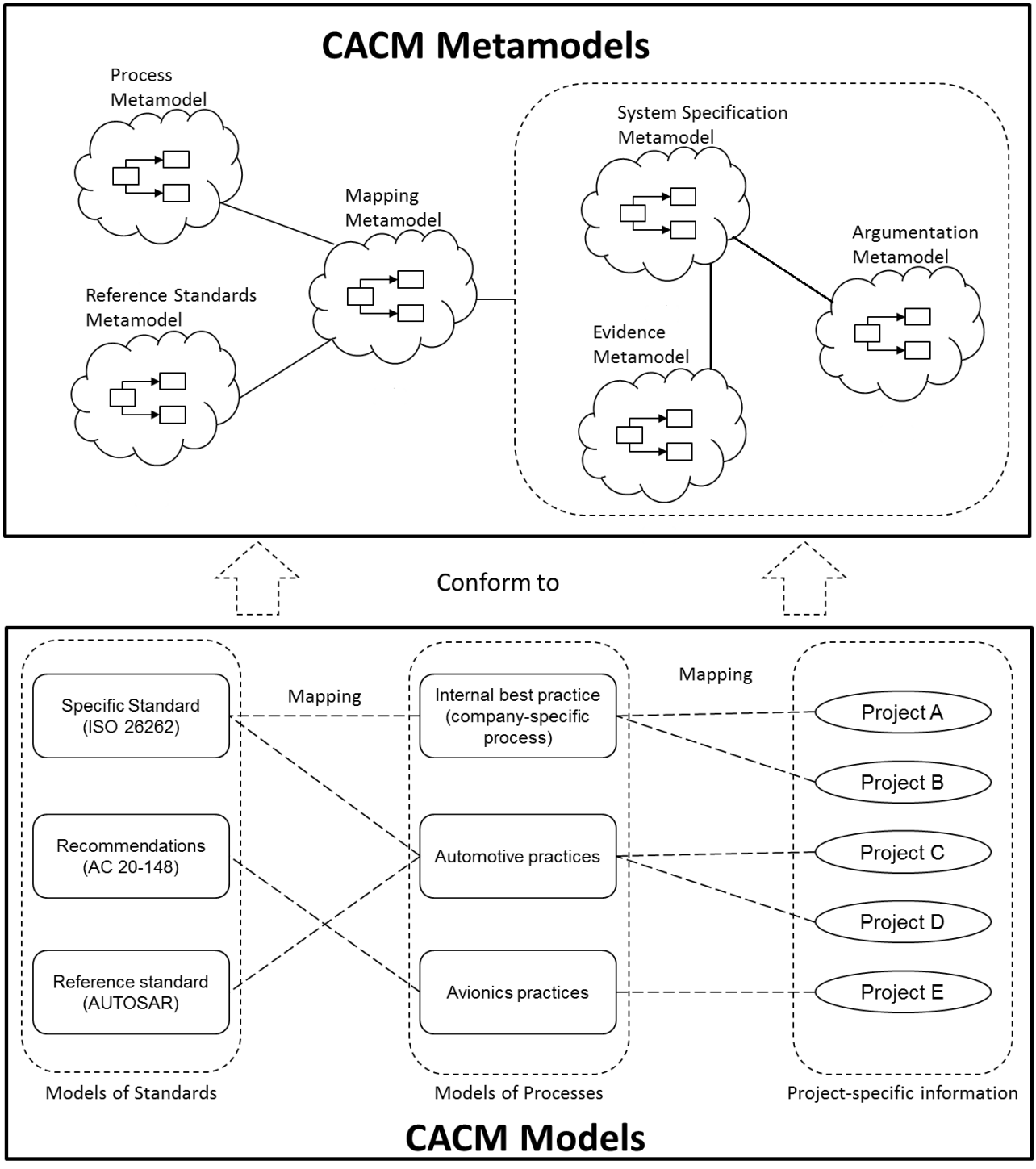


Figure 40. CACM metamodel views

As shown in Figure 40, 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 architectures 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 sub-sections describe the different metamodels specified for the initial version of the CACM from the Conceptual and Implementation perspective, and the maps between both.

4.1 General Metamodel

4.1.1 Scope and Purpose

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

4.1.2 Conceptual General Metamodel

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

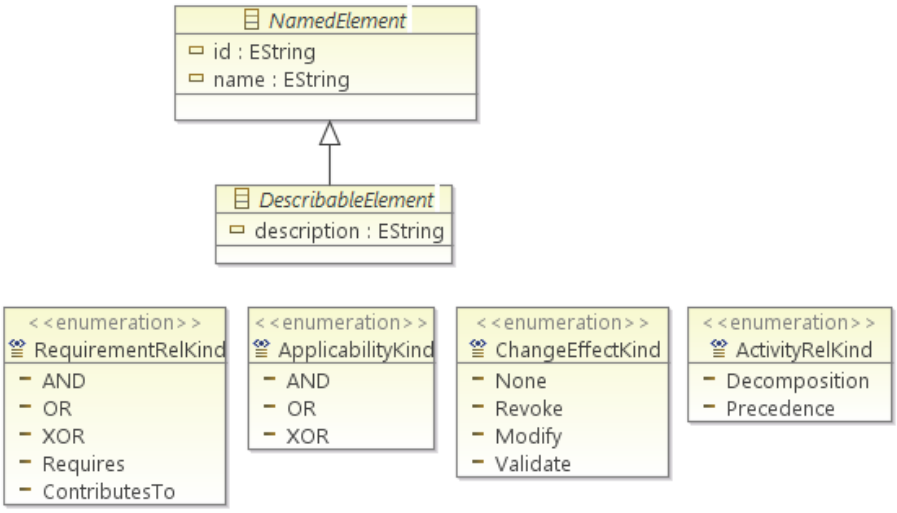


Figure 41. General Metamodel

4.1.2.1 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.

4.1.2.2 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.

4.1.2.3 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.

4.1.2.4 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.

4.1.2.5 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”.

4.1.2.6 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

4.1.3 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 defined in this metamodel, since it belongs to the assurance asset annotated.

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

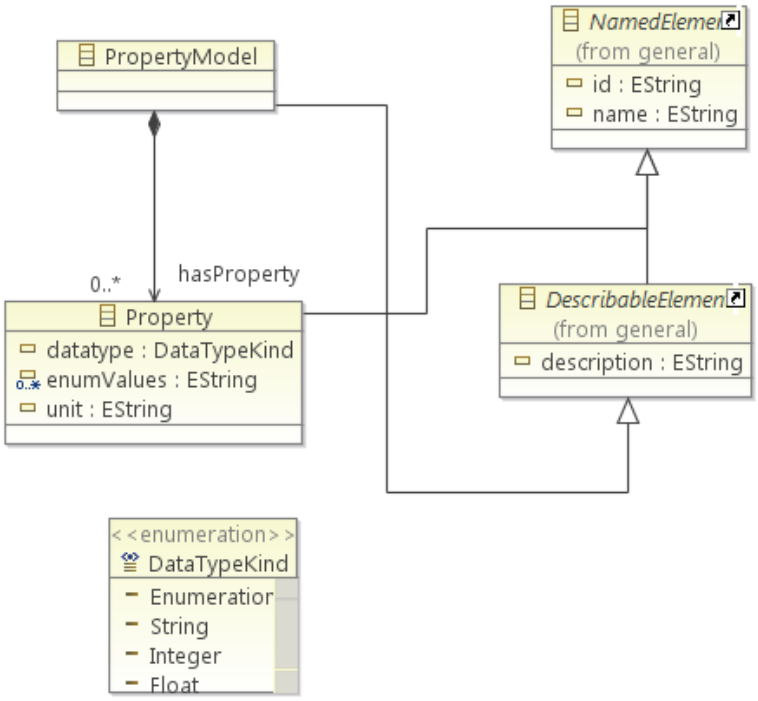


Figure 42. Property Metamodel

4.1.3.1 PropertyModel

This class corresponds to 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.

4.1.3.2 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 defined in this metamodel, since it belongs to the assurance asset annotated. See for instance the Value metaclass in 5.4.3.5.

4.1.3.3 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.

4.2 System Component Metamodel

4.2.1 Scope and Purpose

This section illustrates the System Component MetaModel supporting Architecture-driven assurance (CMMA). The metamodel is a review of the SafeCer metamodel [39] and embeds results from the SEooCMM metamodel [40].

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 [37].

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.

4.2.2 Conceptual Model Definition

Please refer to the D3.3 deliverable [14].

4.3 Assurance Case Metamodel

4.3.1 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) [27]. 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).

4.3.2 Conceptual Model Definition

The metamodel presented here is an extension of the SACM Metamodel [27] from OMG (Object Management Group).

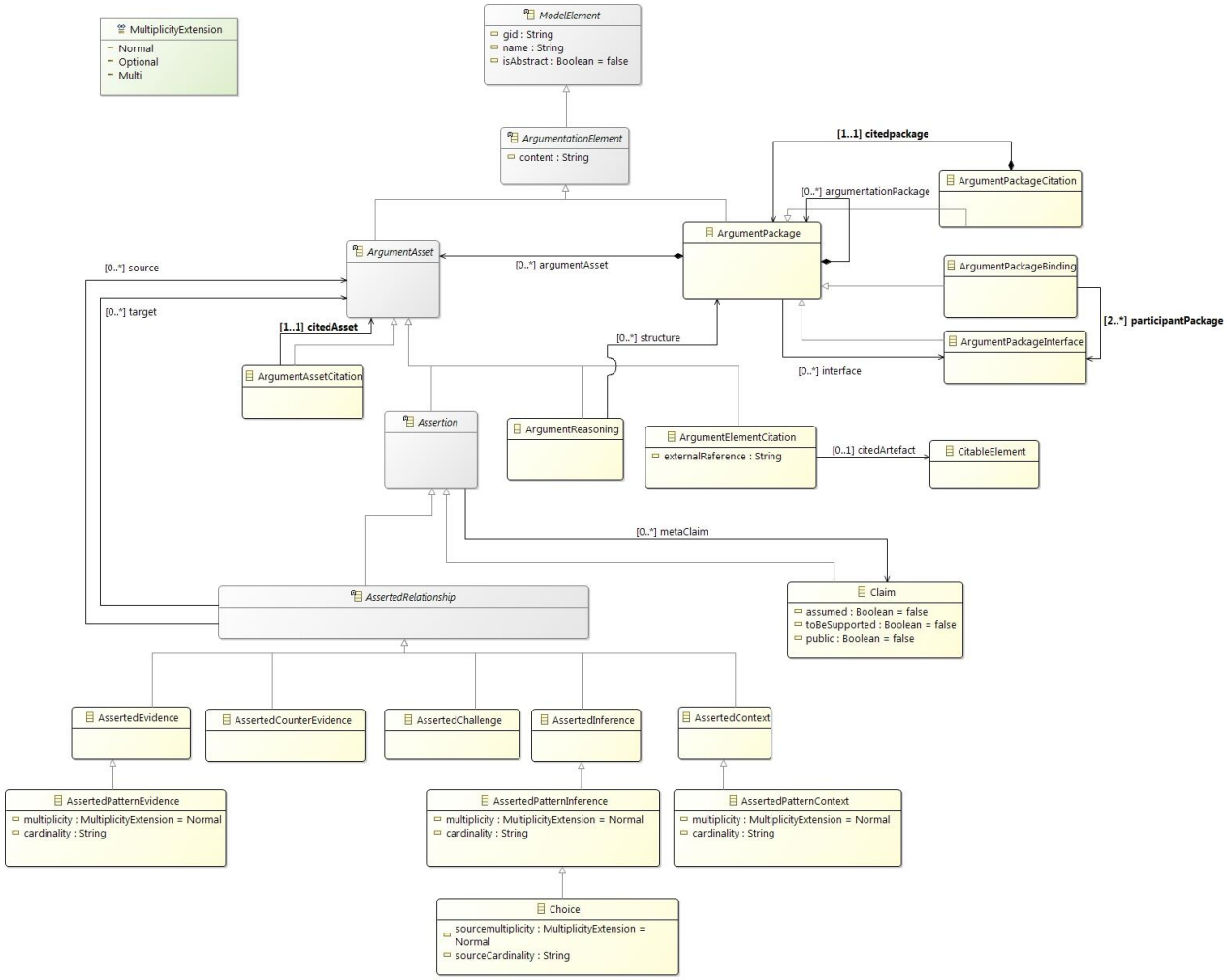


Figure 43. Conceptual Assurance Case Metamodel diagram

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 [27].

4.3.2.1 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

4.3.2.2 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

4.3.2.3 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

4.3.2.4 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

4.3.2.5 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.

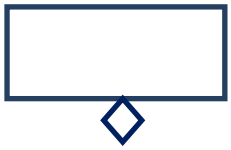
Claims are related with ArgumentElementCitation through the AssertedEvidence relationship. Claims are also related to other claims in a decomposition structure. The AssertedInference relationship is also used to refer to such relationships.

Graphical Notation

assumed=false
toBeSupported=false

assumed=true
toBeSupported=false

assumed=false
toBeSupported=true



assumed=false
toBeSupported=false
public=true

4.3.2.6 CitableElement

It is part of the 4.4.3.2.

4.4 Evidence Management Metamodels

4.4.1 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 [30] and SACM [27], also taking into account some aspects from DAF [31], SPEM [28], and UMA [29]. 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.

4.4.2 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 44) aims at supporting the specification of the main information related to such relationships, which includes information for change impact analysis.

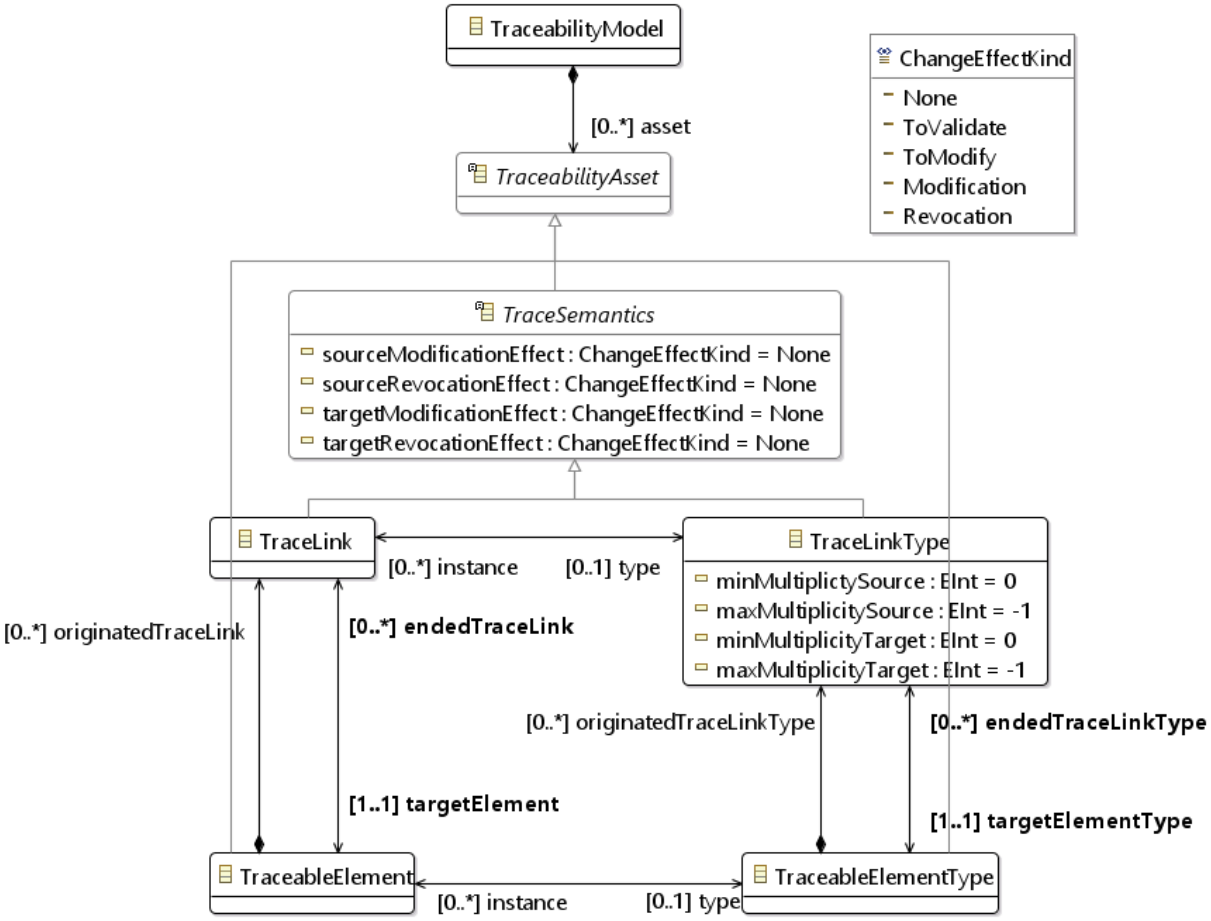


Figure 44. Traceability Metamodel

4.4.2.1 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).

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.

4.4.2.2 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.

4.4.2.3 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.

4.4.2.4 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”.

4.4.2.5 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”.

4.4.2.6 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.

4.4.2.7 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

4.4.3 Conceptual Managed Artefact Metamodel

This metamodel (see Figure 45) corresponds to an adaptation of the OPENCOSS Artefact Metamodel [30]. The ‘Artefact’ has been renamed a ‘ManagedArtefact’, to clearly differentiate the terminology with UMA [29]. ManagedArtefactPackage has been added to support ManagedArtefact grouping, in line with the most recent work on SACM [27].

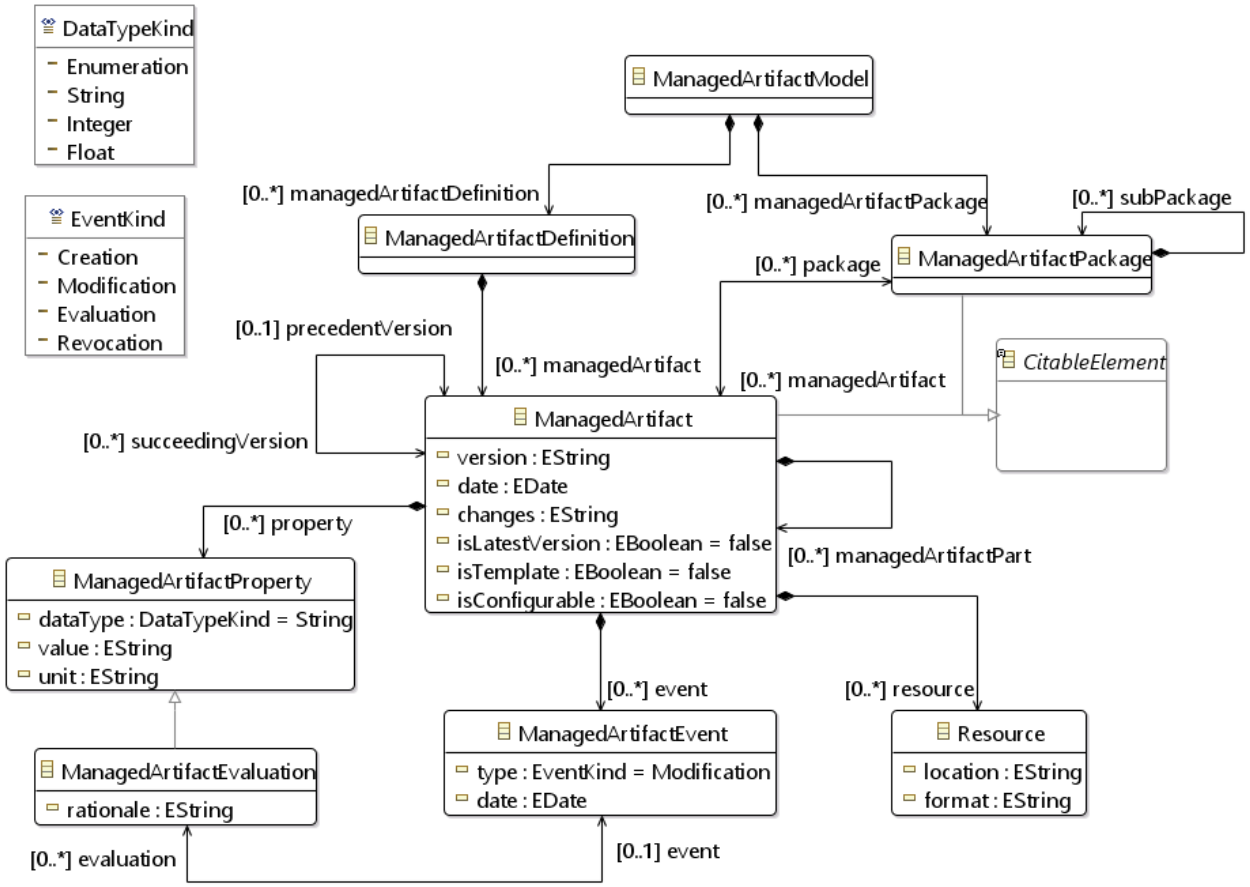


Figure 45. Managed Artifact Metamodel

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.

4.4.3.1 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.

4.4.3.2 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.

4.4.3.3 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.

4.4.3.4 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.

4.4.3.5 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.

4.4.3.6 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.

4.4.3.7 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.

4.4.3.8 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.

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.

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.

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.

4.4.4 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 46).

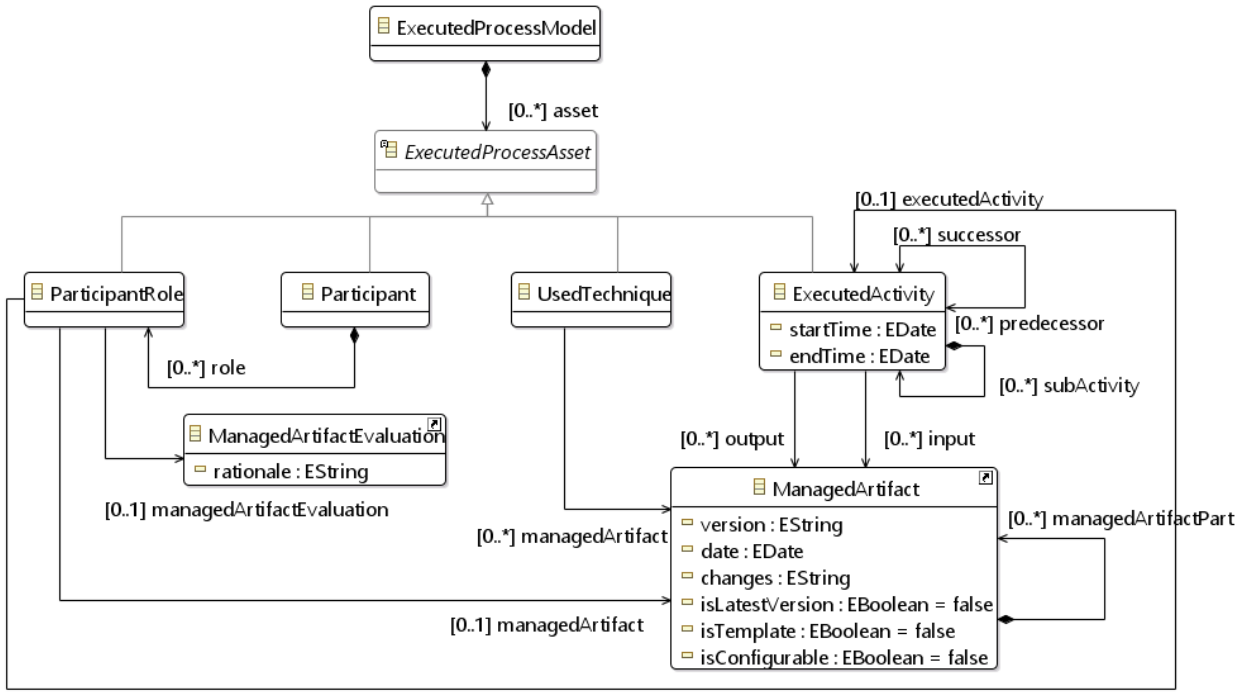


Figure 46. Executed Process Metamodel

4.4.4.1 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.

4.4.4.2 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.

4.4.4.3 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.

4.4.4.4 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.

4.4.4.5 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.

4.4.4.6 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).

4.4.4.7 ManagedArtifact

See definition in 4.4.3.4

4.4.4.8 ManagedArtifactEvaluation

See definition in 4.4.3.7

4.5 Compliance Management Metamodel

4.5.1 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 [32], ISO 26262 [33], DO-178C [34], EN 50126 [35], 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 40 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 47). 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.

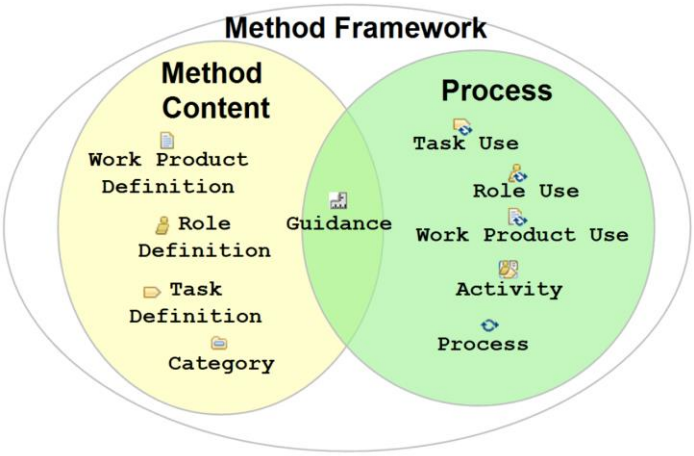


Figure 47. Method Content versus Process in the SPEM 2.0 standard, taken from [28]

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.

4.5.2 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.

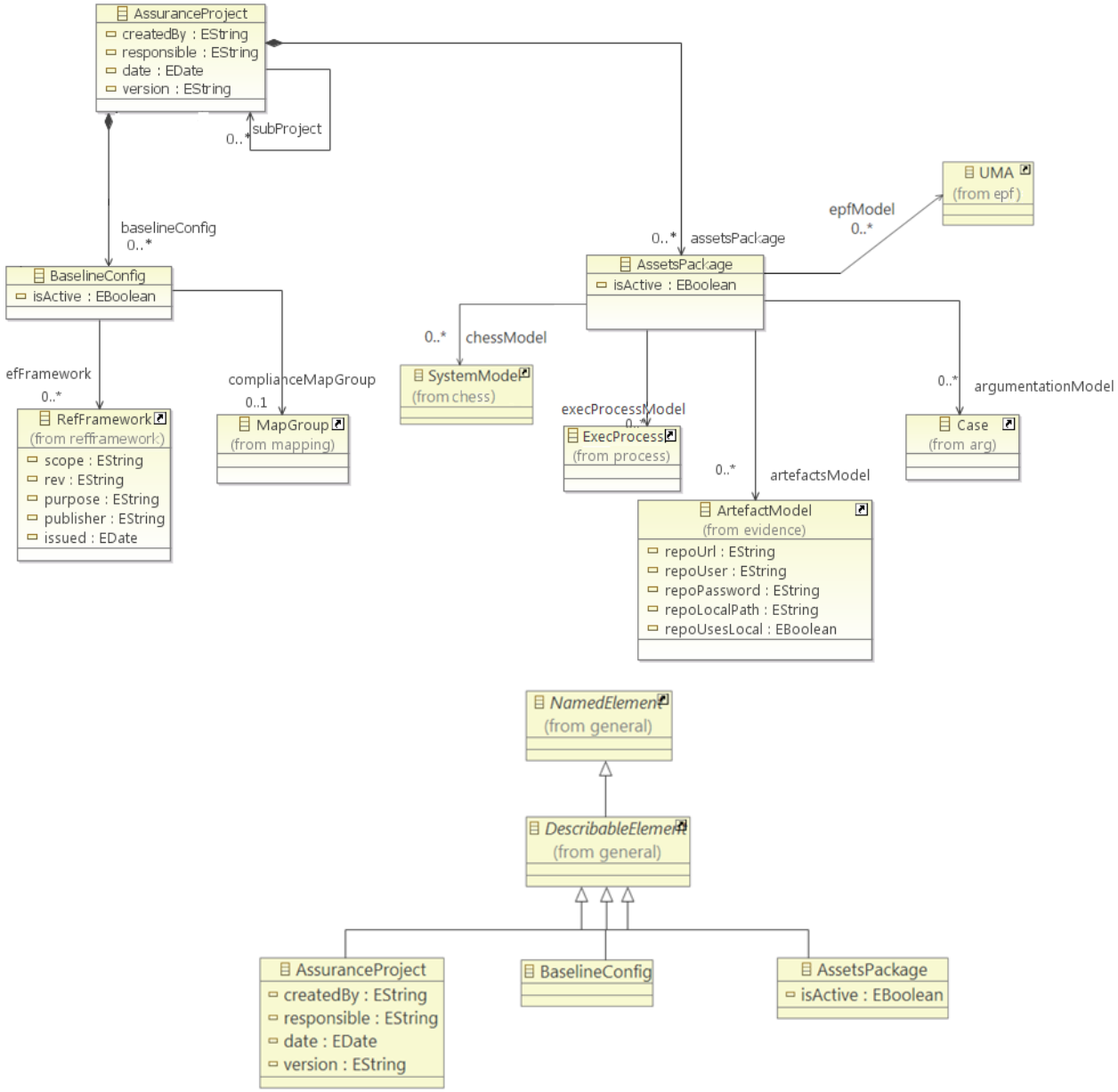


Figure 48. Assurance Project Metamodel

4.5.2.1 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.

4.5.2.2 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.

4.5.2.3 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.

4.5.3 Conceptual Process Definition Metamodel

We do not provide full meta-class description in this document. For further information on UMA, please refer to [5].

As mentioned above, we use UMA for process definition. 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.

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.

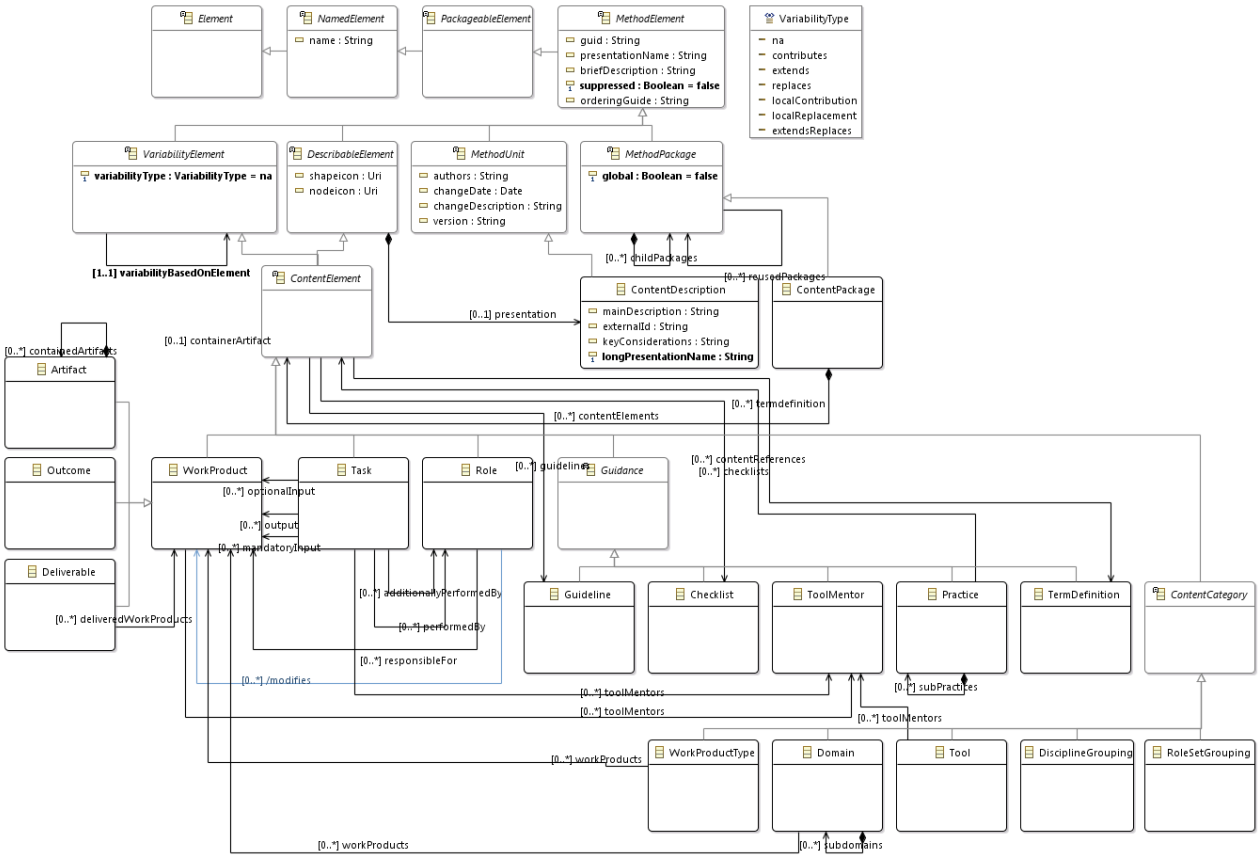


Figure 49. Method Content from the UMA metamodel

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.

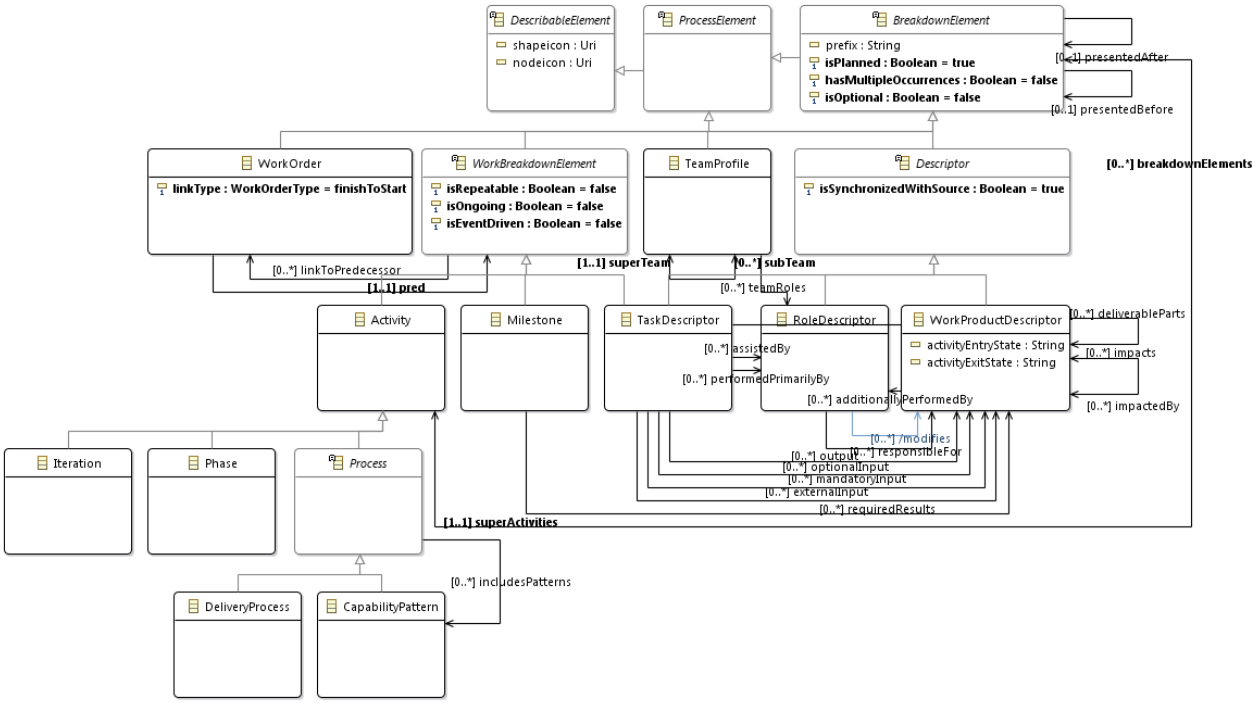


Figure 50. Process from the UMA metamodel

4.5.4 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*) [36]

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

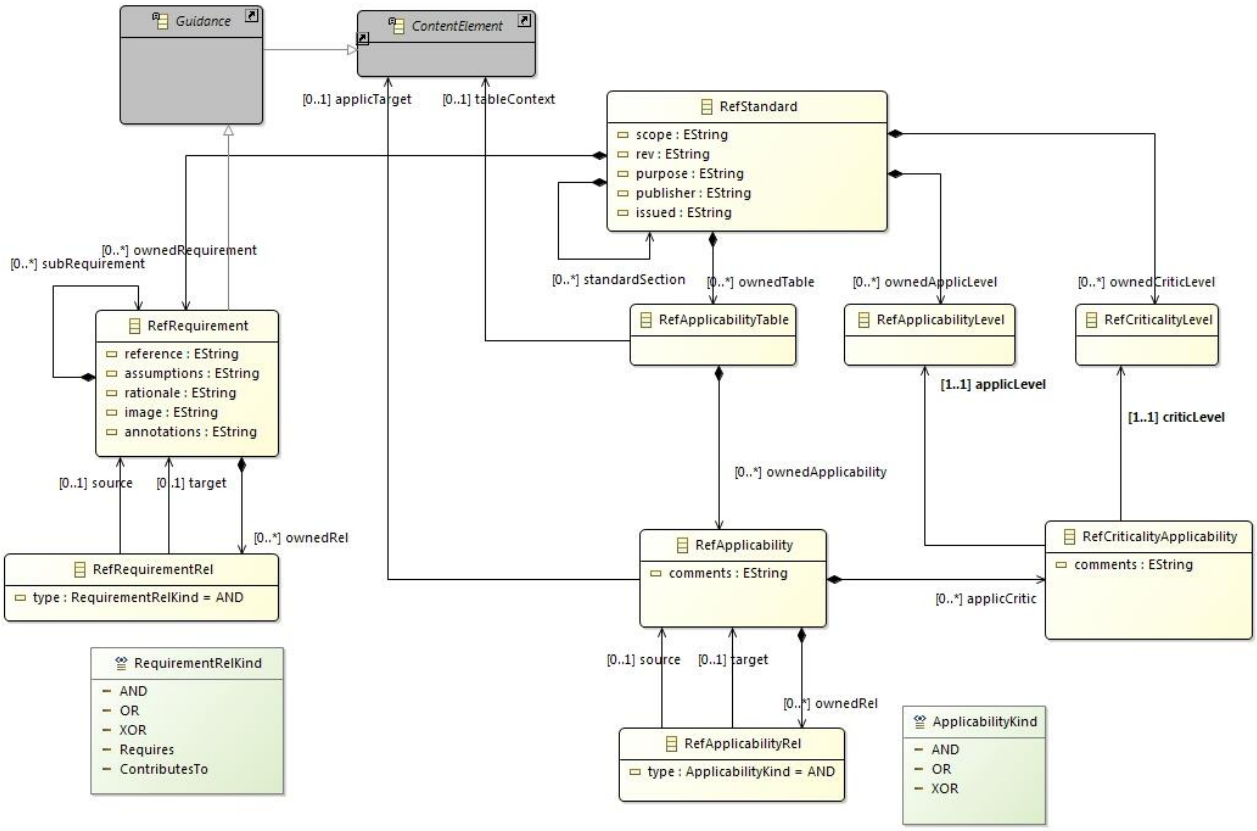


Figure 51. Standard Definition metamodel

4.5.4.1 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 [32]*).

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 [32], ISO 26262 [33], DO-178C [34], EN 50126 [35], 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 [36]*).

4.5.4.2 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.

4.5.4.3 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.

4.5.4.4 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).

4.5.4.5 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).

4.5.4.6 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).

4.5.4.7 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).

4.5.4.8 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).

4.5.4.9 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.

4.5.4.10 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.

4.5.4.11 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.

4.5.4.12 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.

4.5.4.13 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”.

4.5.4.14 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.

4.5.5 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 the Figure below and the subsequent subsections.

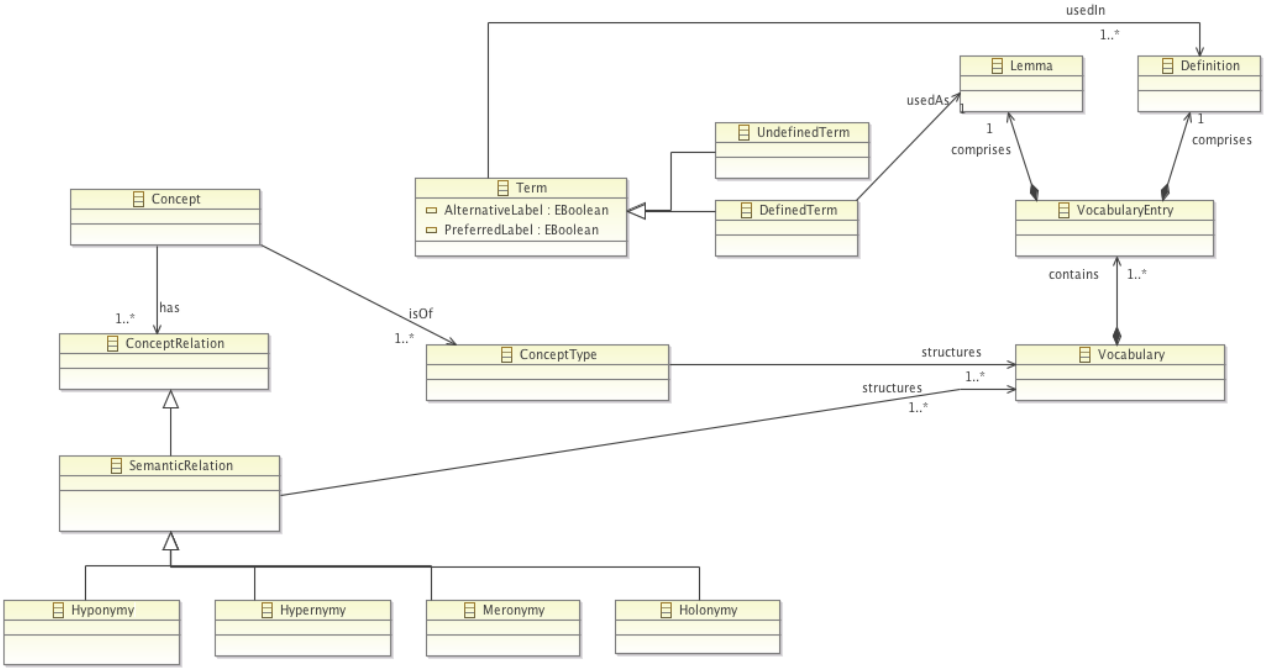


Figure 52. Conceptual Vocabulary Metamodel

4.5.5.1 Concept

Any unit of meaning which can be described or defined by a unique combination of characteristics [38]. A Concept is represented/reified by the Term class and defined by the Definition class.

4.5.5.2 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 [38]. The ConceptTypes provide the basic structure of the Vocabulary in that Terms are arranged according to the ConceptTypes of the Concepts they represent.

4.5.5.3 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.

4.5.5.4 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.

4.5.5.5 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.

4.5.5.6 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.

4.5.5.7 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.

4.5.5.8 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.

4.5.5.9 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.

4.5.5.10 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.

4.5.5.11 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.

4.5.5.12 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.

4.5.5.13 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.

4.5.5.14 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.

4.5.6 Conceptual Mapping Definition Metamodel

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

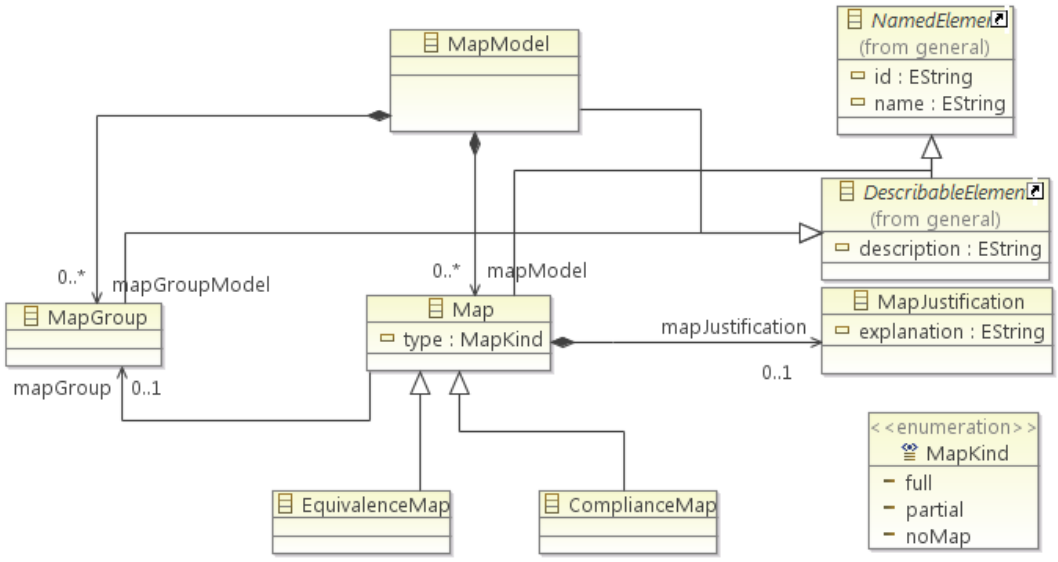


Figure 53. Mapping Definition metamodel

4.5.6.1 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.

4.5.6.2 MapGroup

This class corresponds to a group of Map elements.

Superclass

- *DescribableElement*

Semantics

A Map Group specifies a group of Map elements.

4.5.6.3 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.

4.5.6.4 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.

4.5.6.5 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.

4.5.6.6 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.

4.5.6.7 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.

4.5.6.8 DescribableElement

See definition in 4.1.2.2.

4.5.6.9 NamedElement

See definition in 4.1.2.1.

5. Implementation CACM (*)

5.1 General Metamodel

5.1.1 Scope and Purpose

See 4.1.1.

5.1.2 Implementation General Metamodel

The General conceptual and implementation metamodels are the same.

5.1.3 Implementation Property Metamodel

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

5.2 System Component Metamodel

5.2.1 Scope and Purpose

See 4.2.1.

5.2.2 Implementation Model Definition

5.2.2.1 Modelling out of context

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

5.2.2.1.1 Block Type

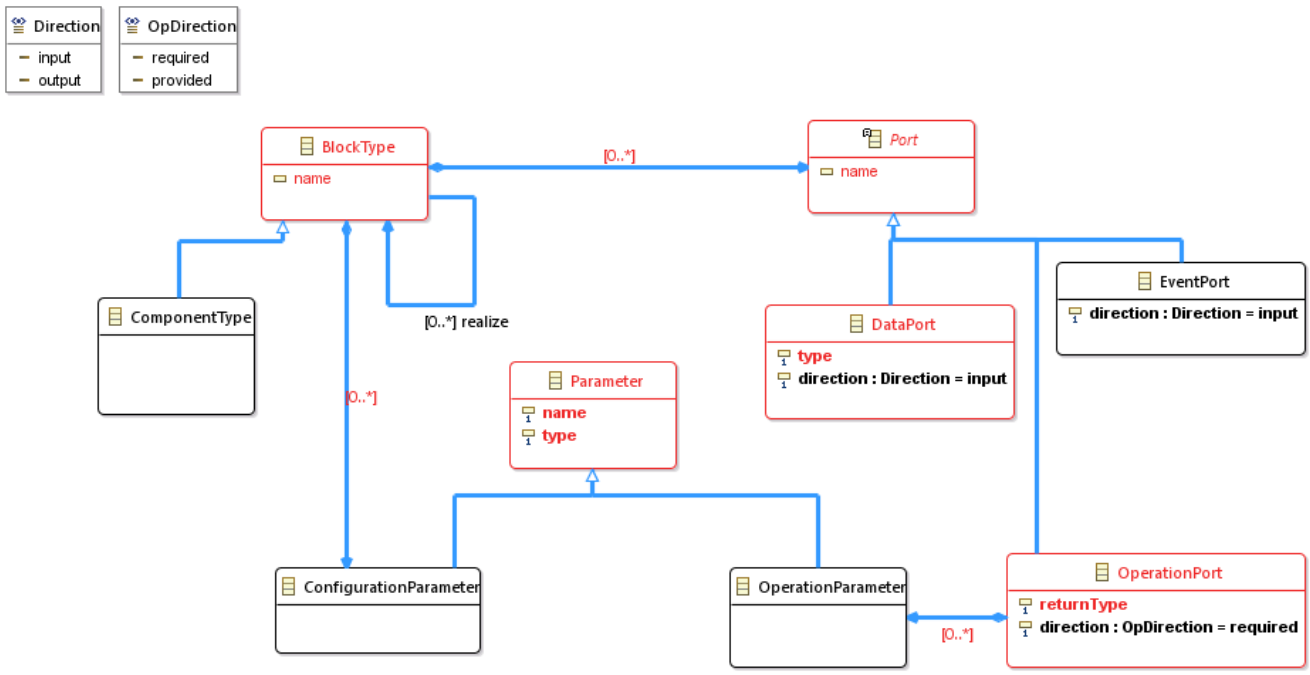


Figure 54. BlockType

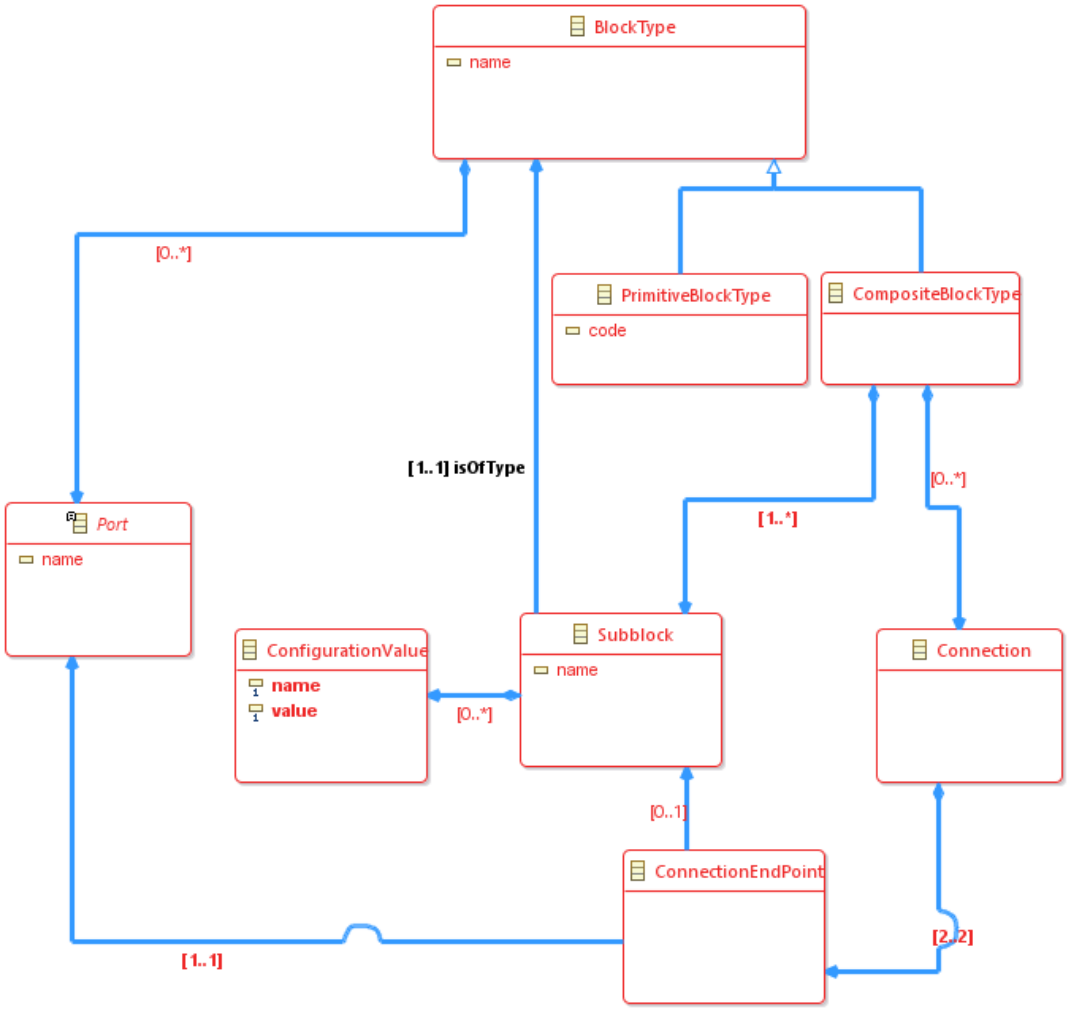


Figure 55. Composite BlockType

A BlockType (Figure 54) 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.

5.2.2.1.2 Port

A Port (Figure 54) 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.

5.2.2.1.3 ConfigurationParameter

Configuration parameters (Figure 54) 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 parameter 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.

5.2.2.1.4 Subblock

Subblock (Figure 55) 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.

5.2.2.1.5 Connection and ConnectionEndPoint

Connection and ConnectionEndPoint (Figure 55) allow to connect Subblocks through the ports defined for the corresponding/typing BlockTypes.

5.2.2.2 Contracts

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

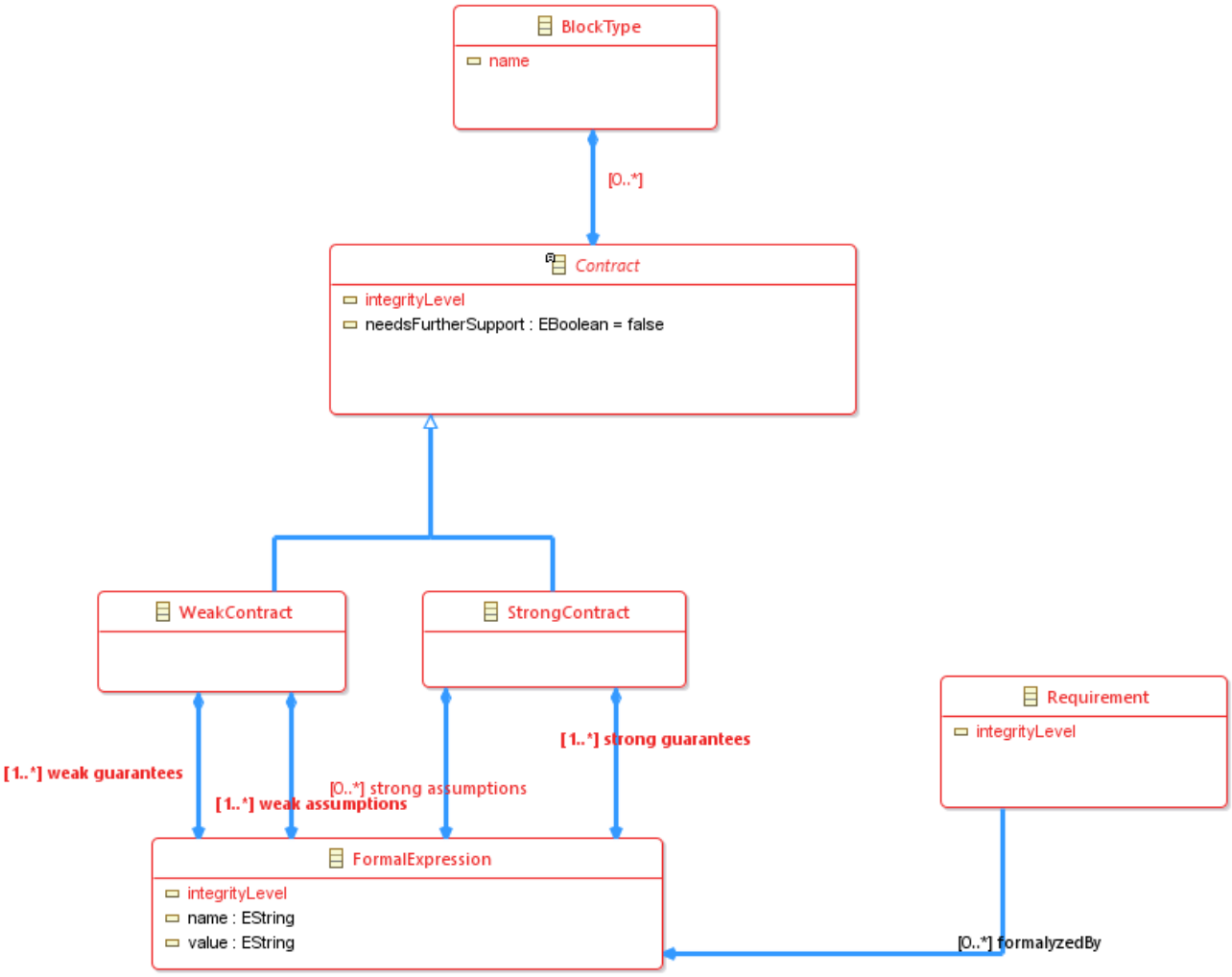


Figure 56. Contract

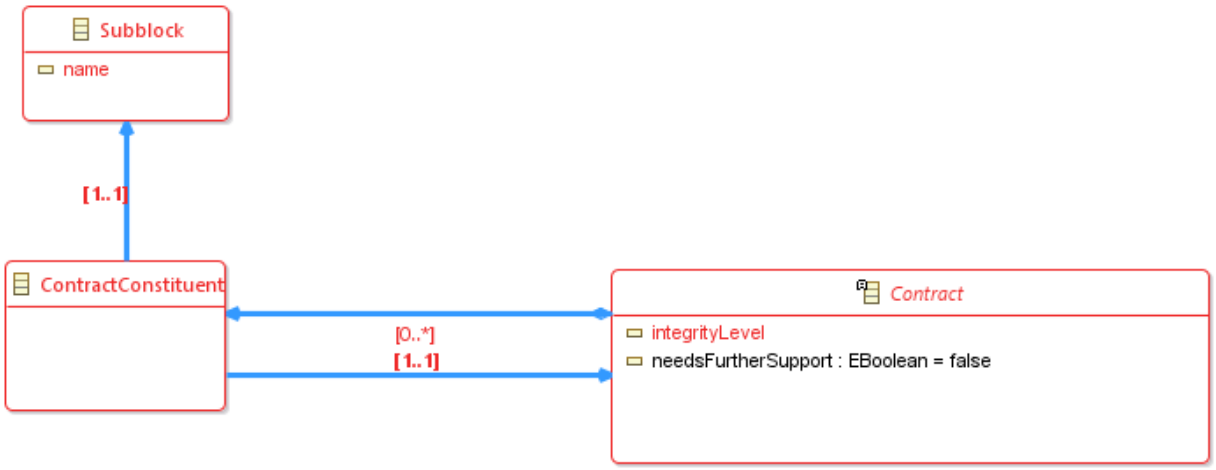


Figure 57. Contract refinement

5.2.2.2.1 Contract

Contracts (Figure 56) 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 integrity level 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.

5.2.2.2.2 FormalProperties

FormalProperty (Figure 56) represents a formalization of a requirement; it appears as assumption or guarantee of a Contract.

5.2.2.2.3 ContractConstituent

ContractConstituent (Figure 57) 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.

5.2.2.3 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.

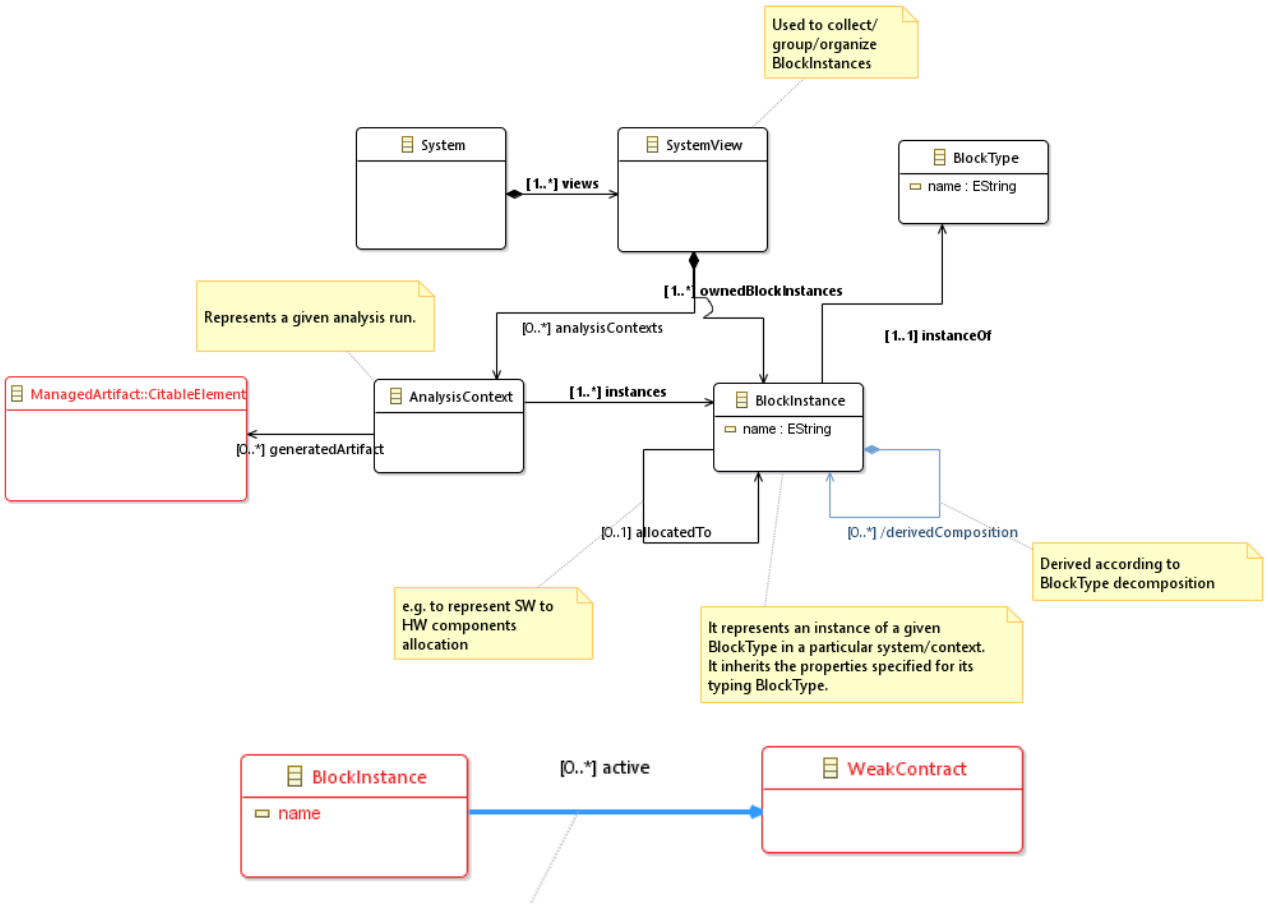


Figure 58. System

5.2.2.3.1 System

A System (Figure 58) represents a given cyber-physical system under design. Hold references to owned block instances through *software* and *platform* association; typically, the latter are created by instantiating a root composite BlockType.

5.2.2.3.2 BlockInstance

BlockInstance (Figure 58) 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:

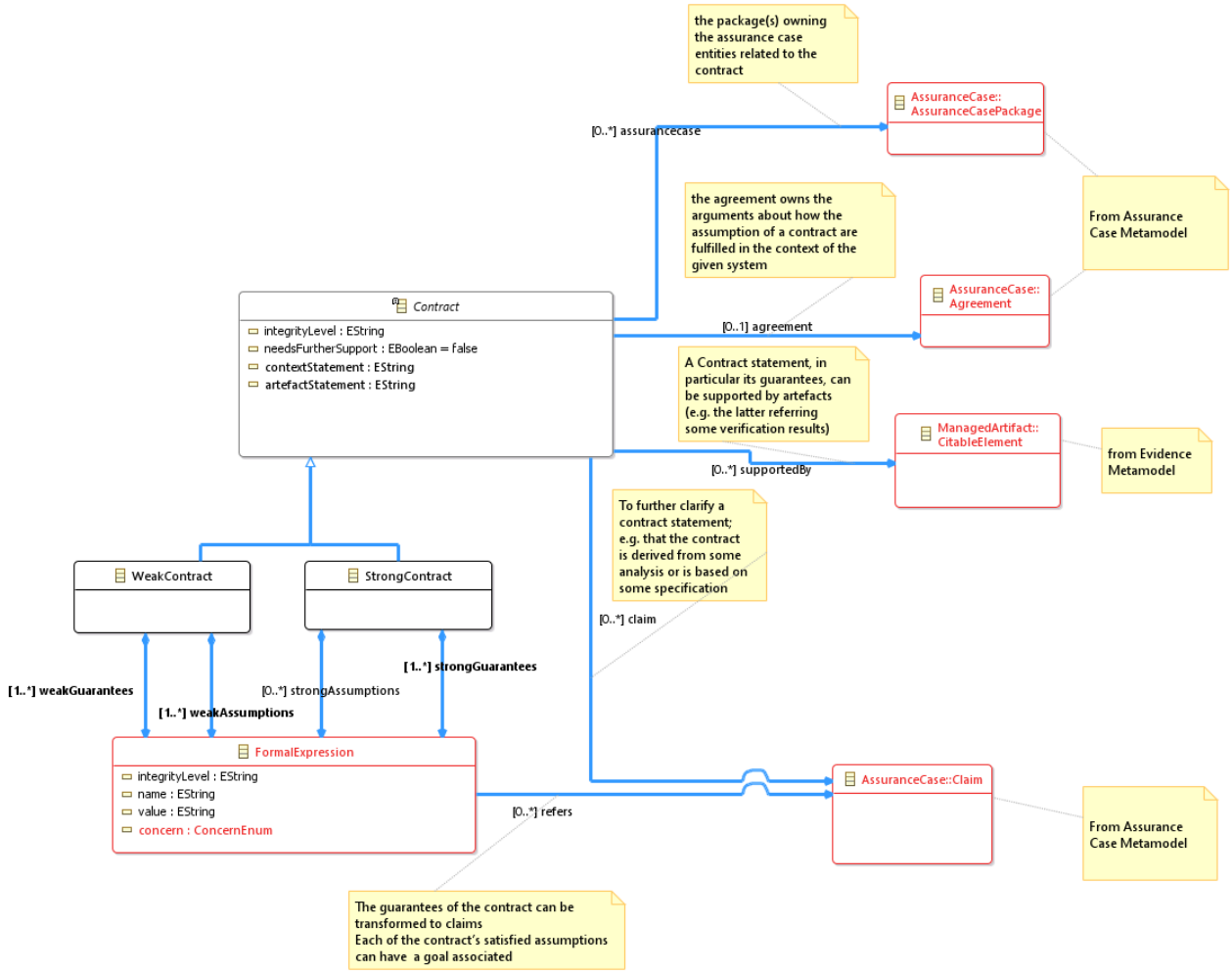
- the StrongContracts associated to the typing BlockType,
- the WeakContracts referred through the *active* relationships.

5.2.2.3.3 AnalysisContext

AnalysisContext (Figure 58) allows to represent a given analysis execution on a (sub)set of block instances. It has trace relationship to the artefacts produced by the corresponding analysis execution.

5.2.2.4 Link to evidence and assurance cases

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



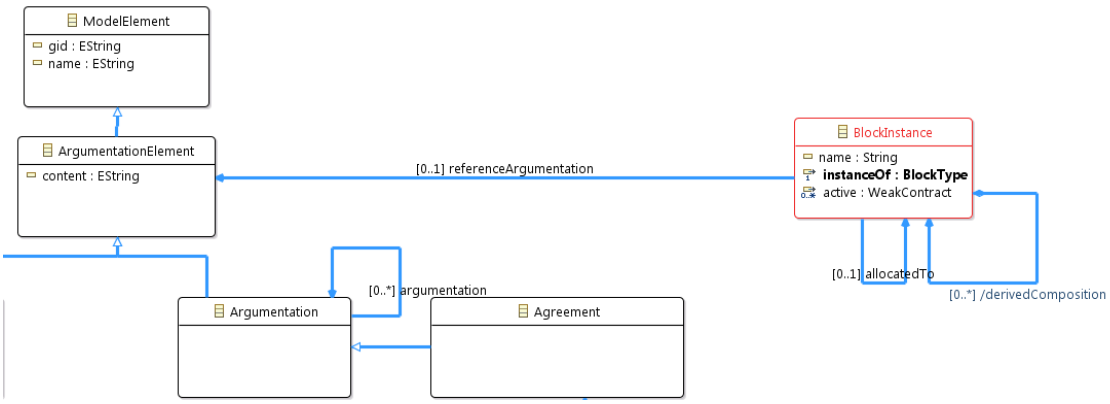


Figure 59. Artefact and assurance-related entities connections

5.2.2.4.1 CitableElement

Imported from AMASS CACM Managed Artifact Metamodel (see 4.4.3.2).

5.2.2.4.2 Claim

Imported from AMASS CACM Assurance Case Metamodel (see 5.3.2.10).

5.2.2.4.3 AssuranceCasePackage

Imported from AMASS CACM Assurance Case Metamodel (see 5.3.2.1).

5.2.2.4.4 Agreement

Imported from AMASS CACM Assurance Case Metamodel (see 5.3.2.2).

5.2.2.4.5 ArgumentationElement

Imported from AMASS CACM Assurance Case Metamodel (see 5.3.2.3).

5.2.2.4.6 BlockInstance

The BlockInstance entity (see 5.2.2.3.2) is extended with the following relationship:

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

5.2.2.4.7 Contract

The Contract entity (see 5.2.2.3.1) 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.

5.2.2.4.8 FormalExpression

The FormalProperty entity (see 5.2.2.2.2) 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.

5.2.2.4.9 Requirement

The Requirement entity (introduced here as a general entity representing system level requirements) is extended with the following relationships:

- supportedBy: SupportArtefact
  - It is not sufficient to establish that the set of formalised properties addressing the requirement is achieved, but also it needs to be established that the set of formalised properties is sufficient to address the requirement. This is for the case that the requirement is addressed by a set of contracts. The idea is to attach such evidence directly to the requirement, on the same level in the generated argument as the satisfaction of the set of related contracts.

5.2.2.5 Failure Behaviour

This part of the metamodel regards the definition of the failure behaviour for a given BlockType. This is a work in progress, in particular for what concerns the identification of connections with the assurance-related entities.

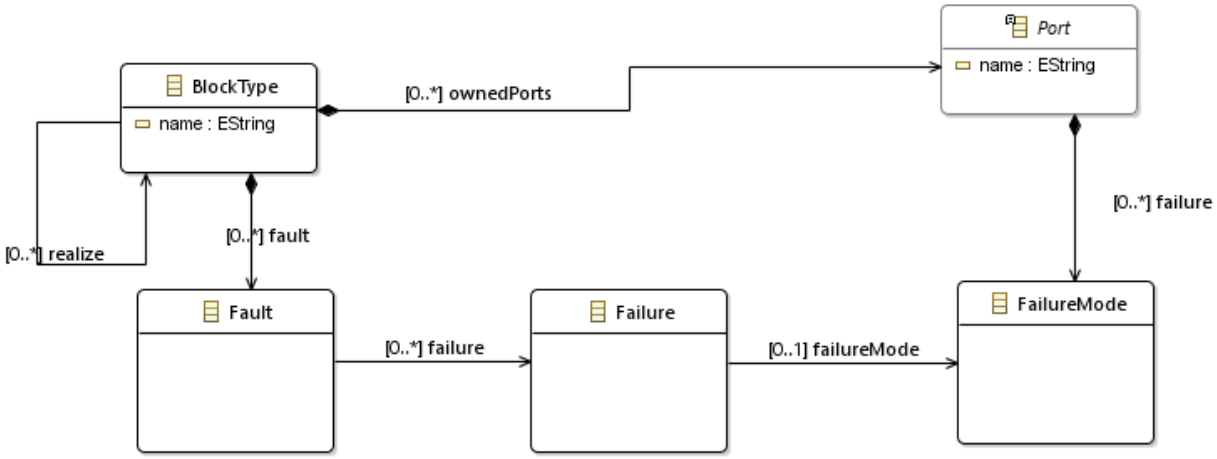


Figure 60. Failure Behaviour

5.3 Assurance Case Metamodel

5.3.1 Scope and Purpose

See 4.3.1.

5.3.2 Implementation Model Definition

The Implementation Assurance Case Metamodel has been split into a series of Metamodel diagrams, which are presented below (Figure 61, Figure 62 and Figure 63).

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 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.

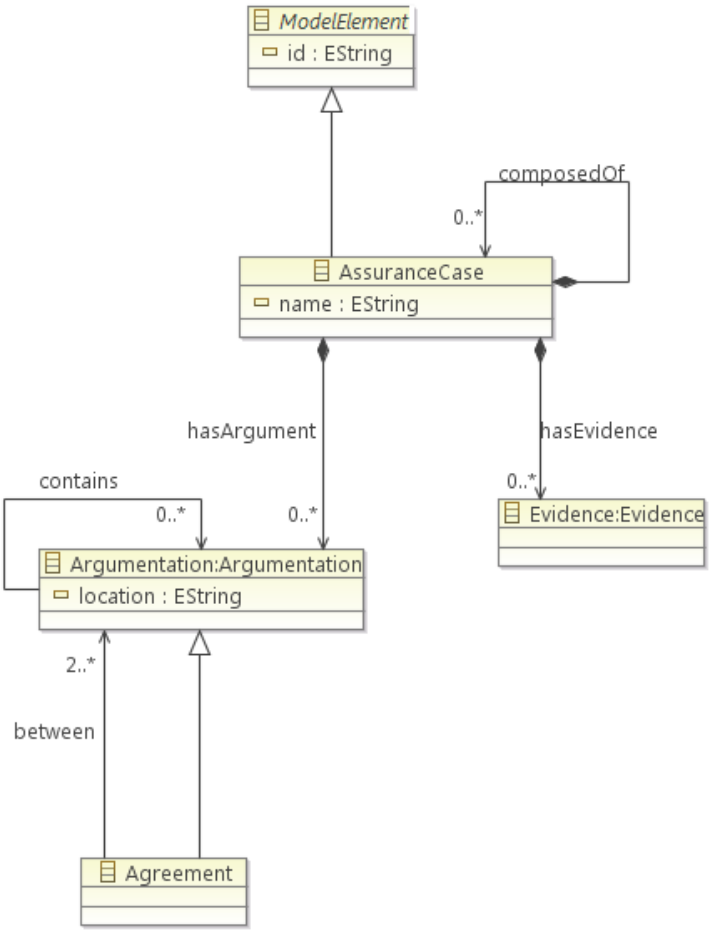


Figure 61. Assurance Case class diagram

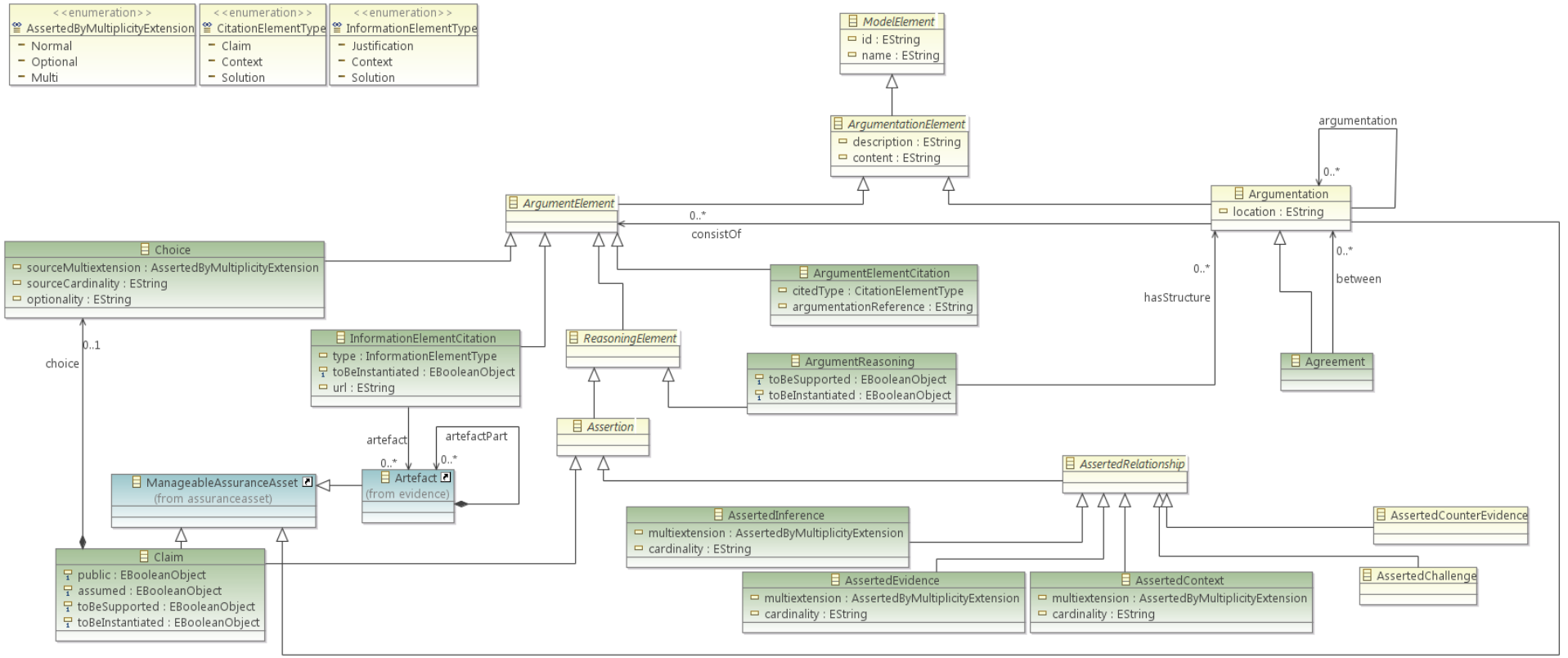


Figure 62. Argumentation Class Diagram

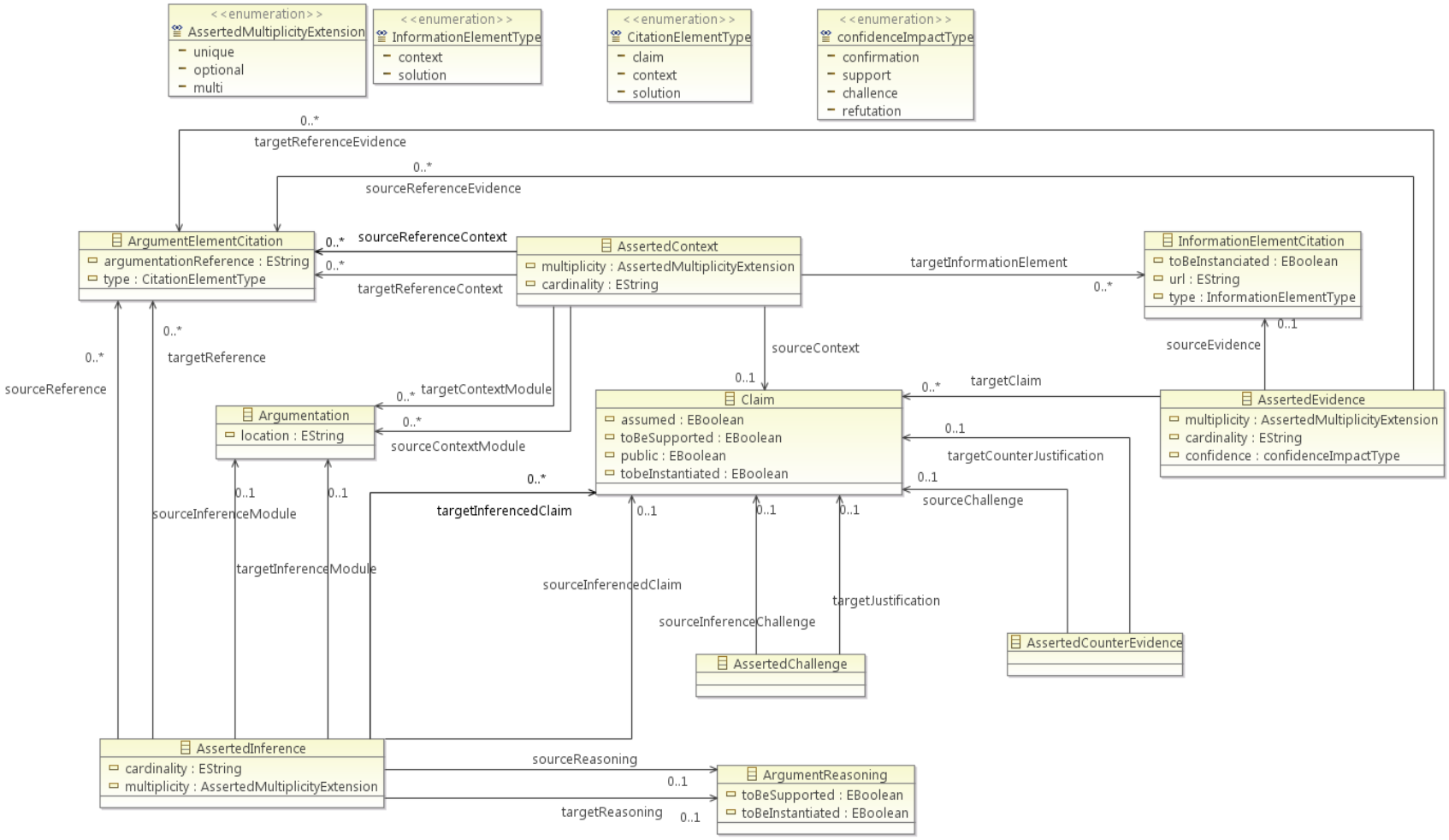


Figure 63. The Relationships view diagram

5.3.2.1 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

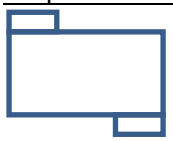
None

5.3.2.2 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



5.3.2.3 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

5.3.2.4 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.

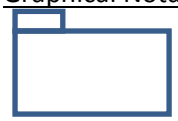
Semantics

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



5.3.2.5 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

5.3.2.6 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

5.3.2.7 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

5.3.2.8 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”

5.3.2.9 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

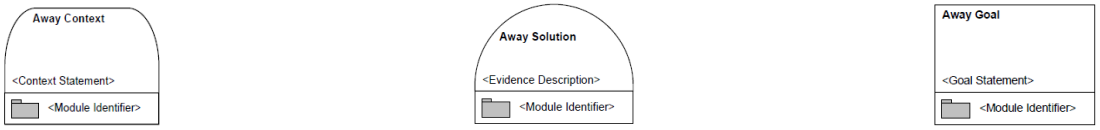
- citesElement:ArgumentElement[0..*]
  References an ArgumentElement within another Argument.

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”

5.3.2.10 Claim

Claims, maps with 4.3.2.5, 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

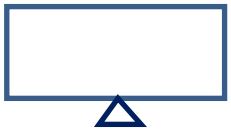
Graphical Notation

assumed=false
toBeSupported=false
toBeInstantiated=false

assumed=true
toBeSupported=false
toBeInstantiated=false

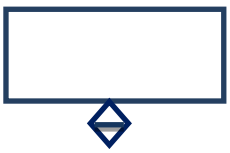
assumed=false
toBeSupported=true
toBeInstantiated=false

assumed=false
toBeSupported=false
toBeInstantiated=true



assumed=false
toBeSupported=false
toBeInstantiated=false
public=true

assumed=false
toBeSupported= true
toBeInstantiated=true



5.3.2.11 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.

5.3.2.12 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

5.3.2.13 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.

5.3.2.14 AssertedInference

The AssertedInference, joined with AssertedRelationship maps with 4.3.2.2, 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

5.3.2.15 Choice

This class, maps with 4.3.2.3, 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

5.3.2.16 AssertedEvidence

The AssertedEvidence, joined with AssertedRelationship maps with 4.3.2.1, 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

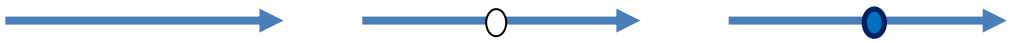
context AssertedEvidence

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



5.3.2.17 AssertedContext

The AssertedContext, maps with 4.3.2.4, 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

5.3.2.18 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

5.3.2.19 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))

5.3.2.20 ManageableAssuranceAsset

See definition in 5.4.3.3.

5.3.2.21 Artefact

Maps with 4.3.2.6. See its definition in 5.4.3.3.

5.4 Evidence Management Metamodels

5.4.1 Scope and Purpose

See 4.4.1.

5.4.2 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 5.4.3.6

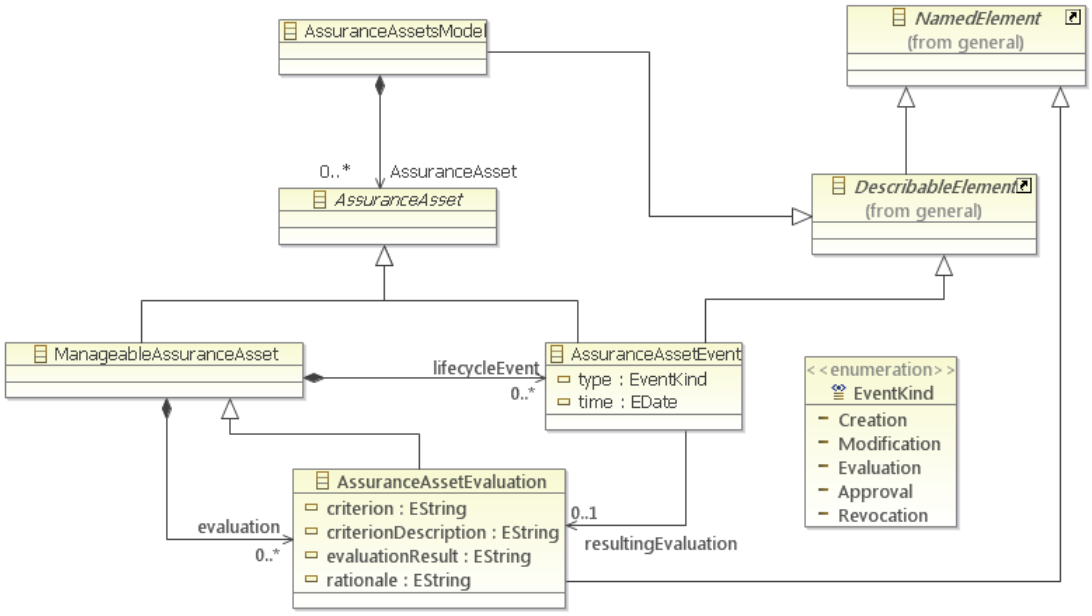


Figure 64. Assurance Asset Metamodel

5.4.2.1 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.

5.4.2.2 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).

5.4.2.3 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.

5.4.2.4 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.

5.4.2.5 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.

5.4.2.6 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.

5.4.3 Implementation Managed Artefact Metamodel

From the implementation perspective, the metamodel for Artefacts is the OPENCOSS Artefact Metamodel [30] without any of the adaptations mentioned in 4.4.3

The class diagram for the Artefact Model is presented in the Figures below.

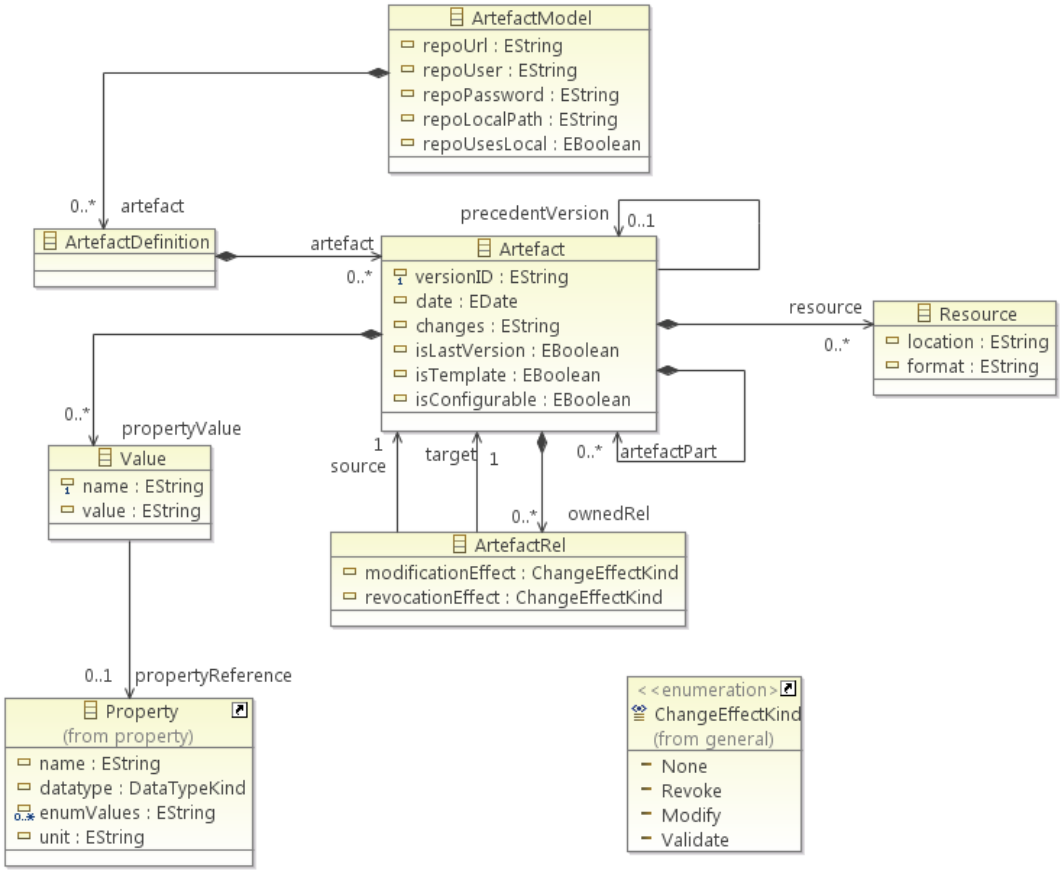


Figure 65. Artefact Metamodel (Part 1: Core Model Elements)

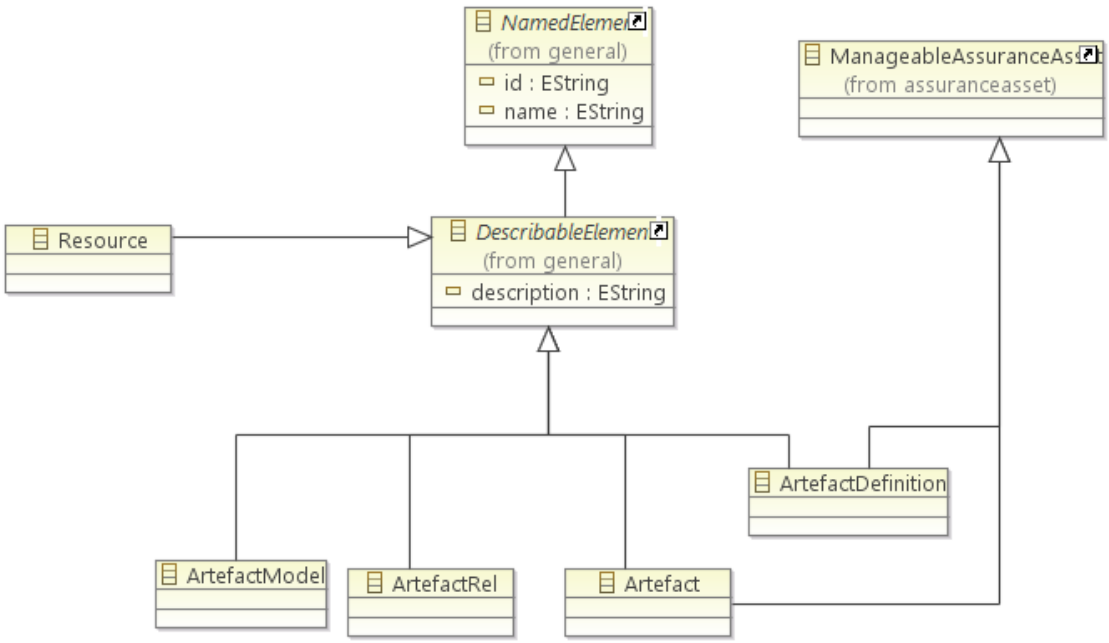


Figure 66. Artefact Metamodel (Part 1: Core Model Elements

5.4.3.1 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.

5.4.3.2 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.

5.4.3.3 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.

5.4.3.4 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.

5.4.3.5 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.

5.4.3.6 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.

5.4.3.7 Property

See definition in 4.1.3.2.

5.4.3.8 ChangeEffectKind

See definition in 4.1.2.4.

5.4.3.9 DescribableElement

See definition in 4.1.2.2.

5.4.3.10 NamedElement

See definition in 4.1.2.1.

5.4.4 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.

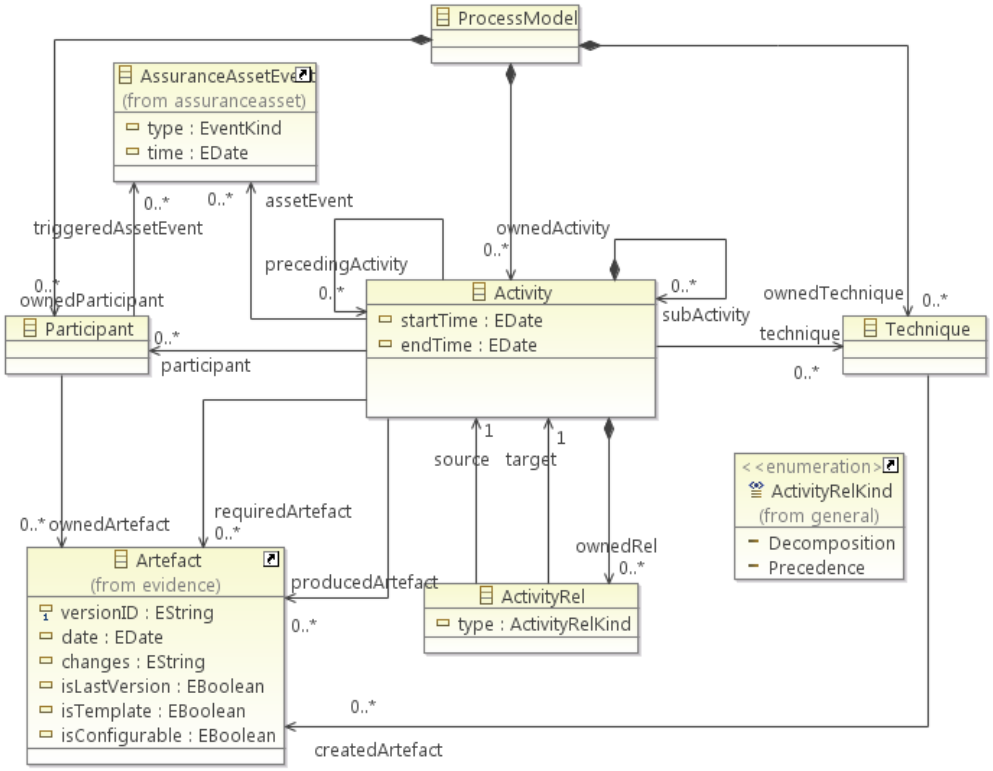


Figure 67. Process Metamodel (Part 1: Core Model Elements)

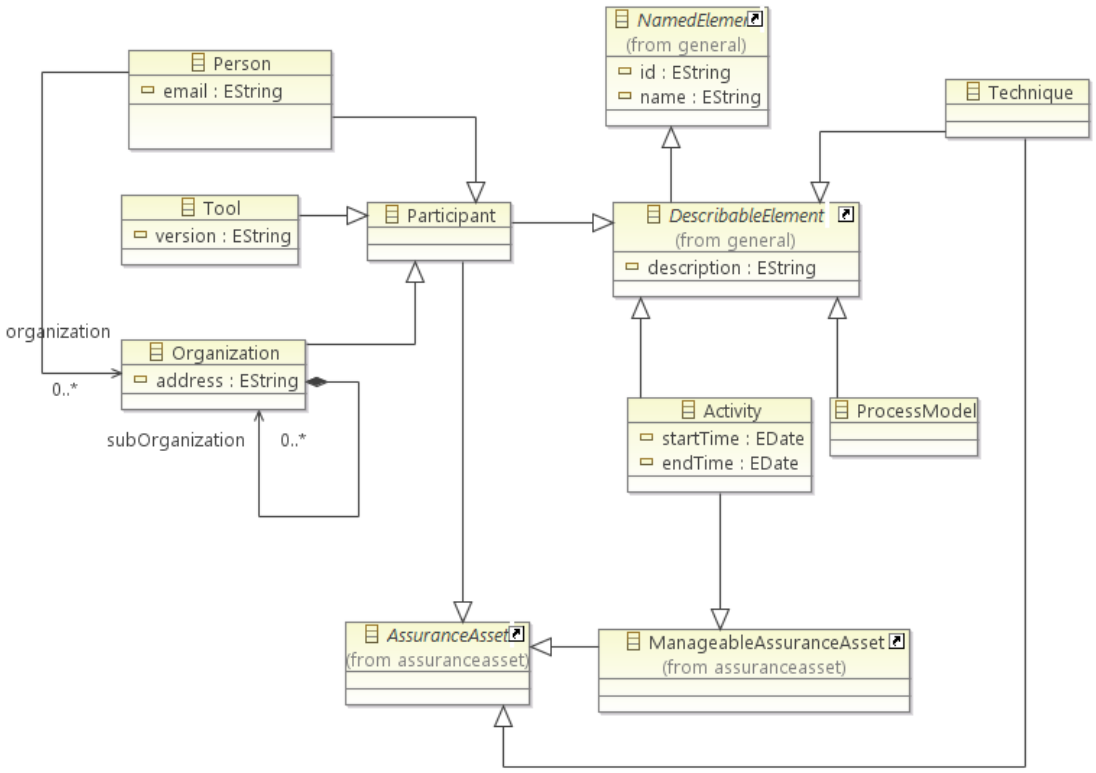


Figure 68. Process Metamodel (Part 2: Inheritance Relationships)

5.4.4.1 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 5.4.2.2 by inheritance.
- ownedParticipant: *Participant* [0..*]
  The set of Participants that belongs to the Process Model. This element is an 5.4.2.2 by inheritance.
- ownedTechnique: *Technique* [0..*]
  The set of Techniques that belongs to the Process Model. This element is an 5.4.2.2 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.

5.4.4.2 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.

5.4.4.3 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.

5.4.4.4 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.

5.4.4.5 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.

5.4.4.6 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.

5.4.4.7 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.

5.4.4.8 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.

5.4.4.9 Artefact

See definition in 5.4.3.3

5.4.4.10 ActivityRelKind

See definition in 4.1.2.3.

5.4.4.11 AssuranceAssetEvent

See definition in 5.4.2.5.

5.4.4.12 ManageableAssuranceAsset

See definition in 5.4.2.3.

5.4.4.13 DescribableElement

See definition in 4.1.2.2.

5.4.4.14 NamedElement

See definition in 4.1.2.1.

5.5 Compliance Management Metamodel

5.5.1 Scope and Purpose

See 4.5.1

5.5.2 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 5.5.2.2.

The class diagram for this Metamodel is presented in the figure below and these differences will be explained in the following subsections.

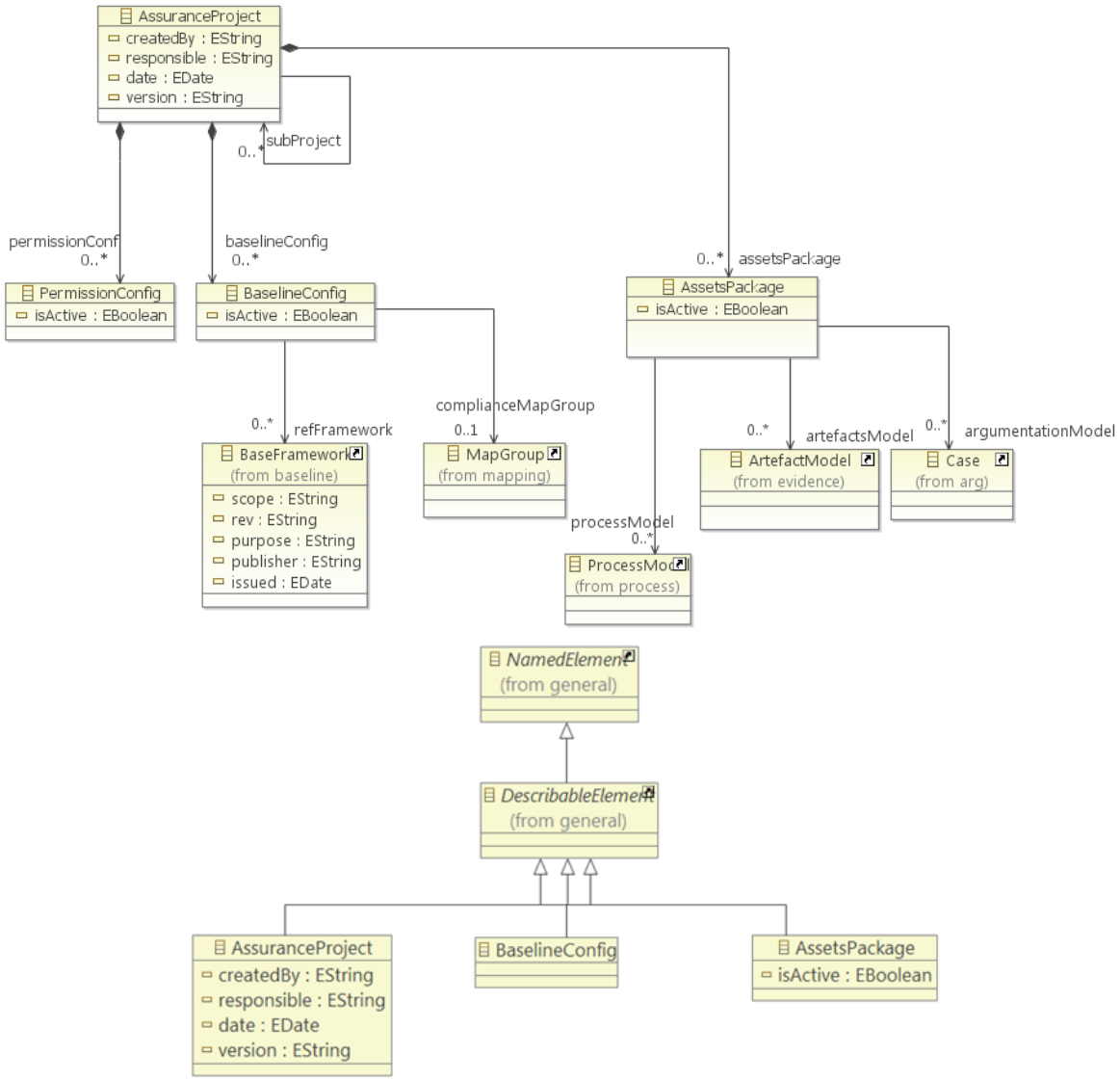


Figure 69. Assurance Project Metamodel

5.5.2.1 AssuranceProject

This class maps with 4.5.2.1.
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.

5.5.2.2 PermissionConfig

This class corresponds to 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 OPENCOSS 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 OPENCOSS functionality and data.

5.5.2.3 BaselineConfig

This class maps with 4.5.2.2.

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?).

5.5.2.4 AssetPackage

This class maps with 4.5.2.3.

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.

5.5.2.5 BaseFramework

See definition in 5.5.5.1.

5.5.3 Implementation Process Definition Metamodel

Please refer to the D6.3 deliverable [17].

5.5.4 Implementation Standard Definition Metamodel

The Metamodel for the implementation of 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 70 and Figure 71 below. In the following subsections, we define the model elements.

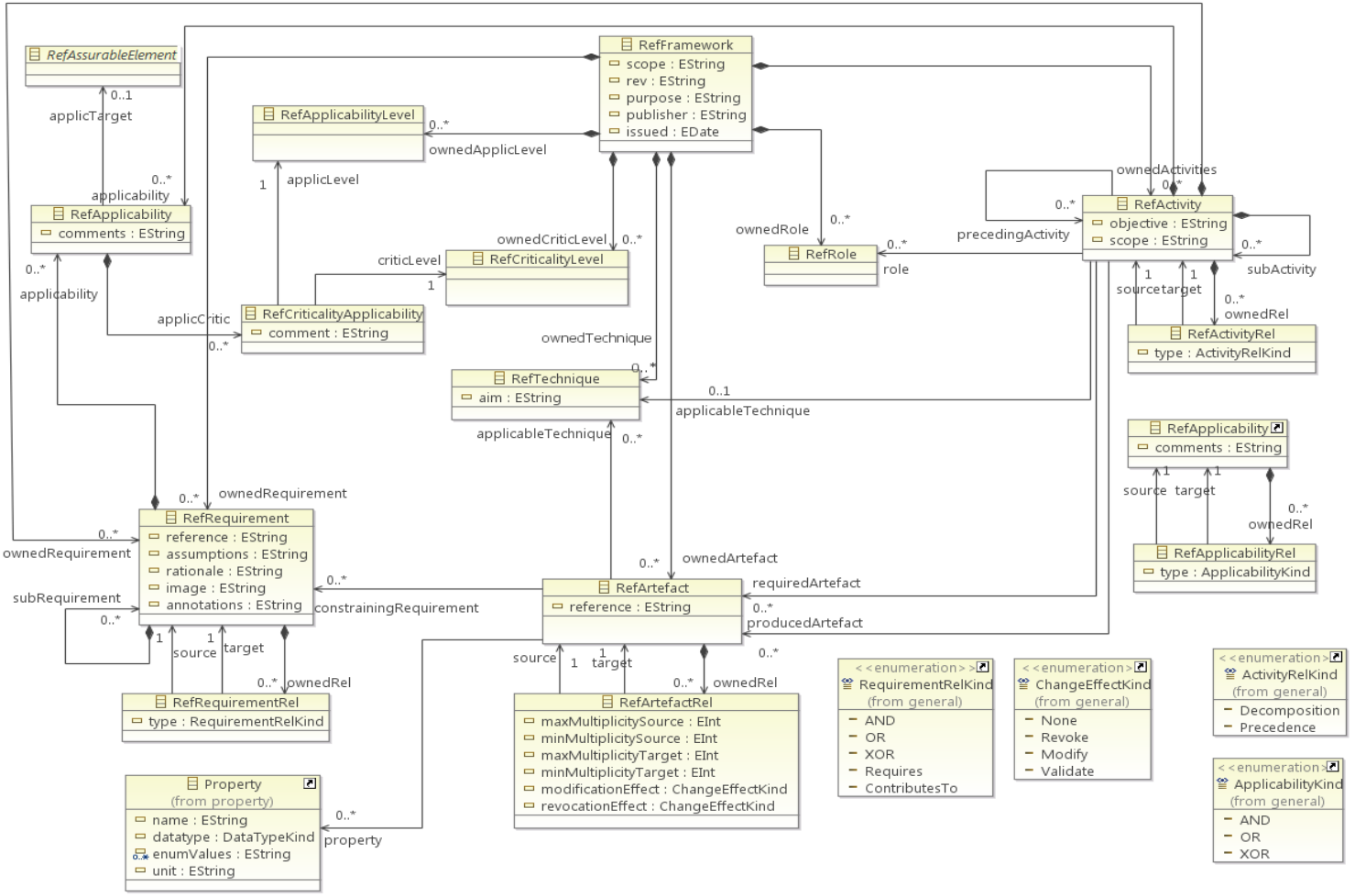


Figure 70. Reference Assurance Framework Metamodel (Part 1: Core Model Elements)

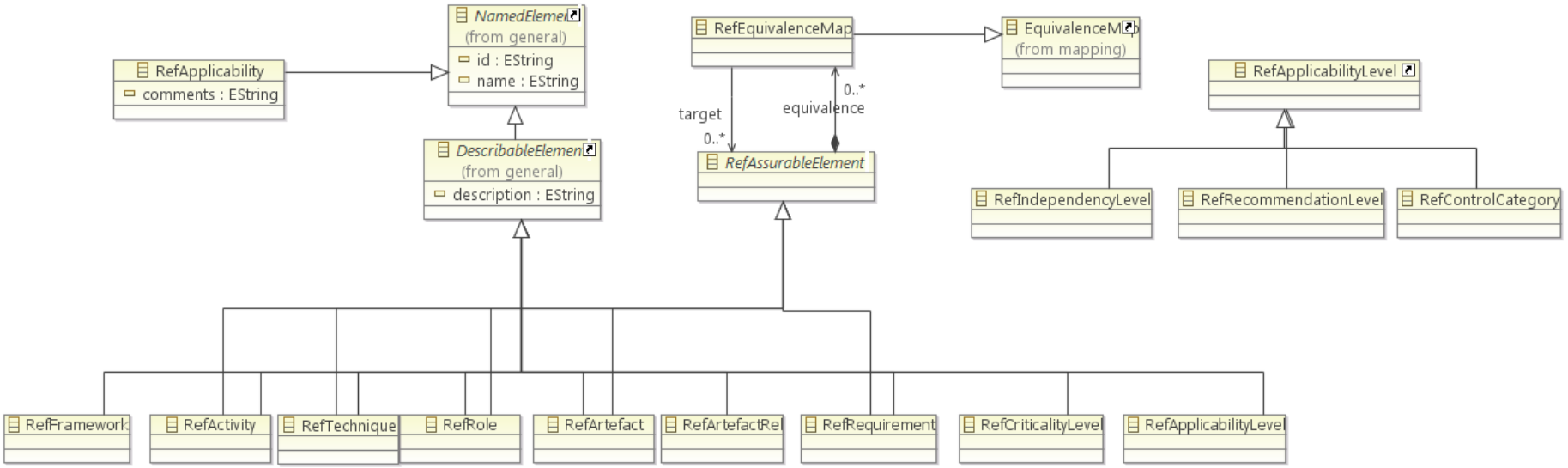


Figure 71. Reference Assurance Framework Metamodel (Part 2: Inheritance Relationships)

5.5.4.1 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*) [36].

5.5.4.2 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

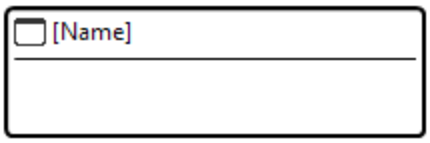
Relationships

- 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):

5.5.4.3 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.

5.5.4.4 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

5.5.4.5 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

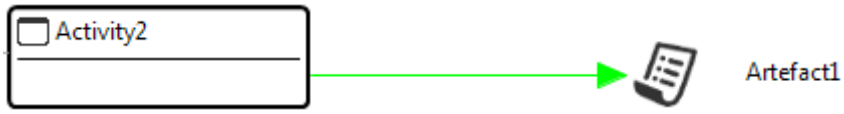
- ownedRel: *RefArtefactRel* [0..*]
  The reference artefact relationships owned by a reference artefact.
- property: *Property* [0..*]
  The reference artefact properties corresponding to the actual artefact can have property values.
- constrainingRequirement: *RefRequirement* [0..*]
  The requirements 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:

5.5.4.6 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)

5.5.4.7 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.

5.5.4.8 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.

5.5.4.9 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.

5.5.4.10 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).

5.5.4.11 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).

5.5.4.12 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).

5.5.4.13 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).

5.5.4.14 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).

5.5.4.15 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.

5.5.4.16 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.

5.5.4.17 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.

5.5.4.18 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.

5.5.4.19 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.

5.5.4.20 ActivityRelKind (enumeration)

See definition in 4.1.2.3.

5.5.4.21 ChangeEffectKind (enumeration)

See definition in 4.1.2.4.

5.5.4.22 RequirementRelKind (enumeration)

See definition in 4.1.2.5.

5.5.4.23 ApplicabilityKind (enumeration)

See definition in 4.1.2.6.

5.5.4.24 Property

See definition in 4.1.3.2.

5.5.4.25 EquivalenceMap

See definition in 4.5.6.4.

5.5.4.26 DescribableElement

See definition in 4.1.2.2.

5.5.4.27 NamedElement

See definition in 4.1.2.1.

5.5.5 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 subsections.

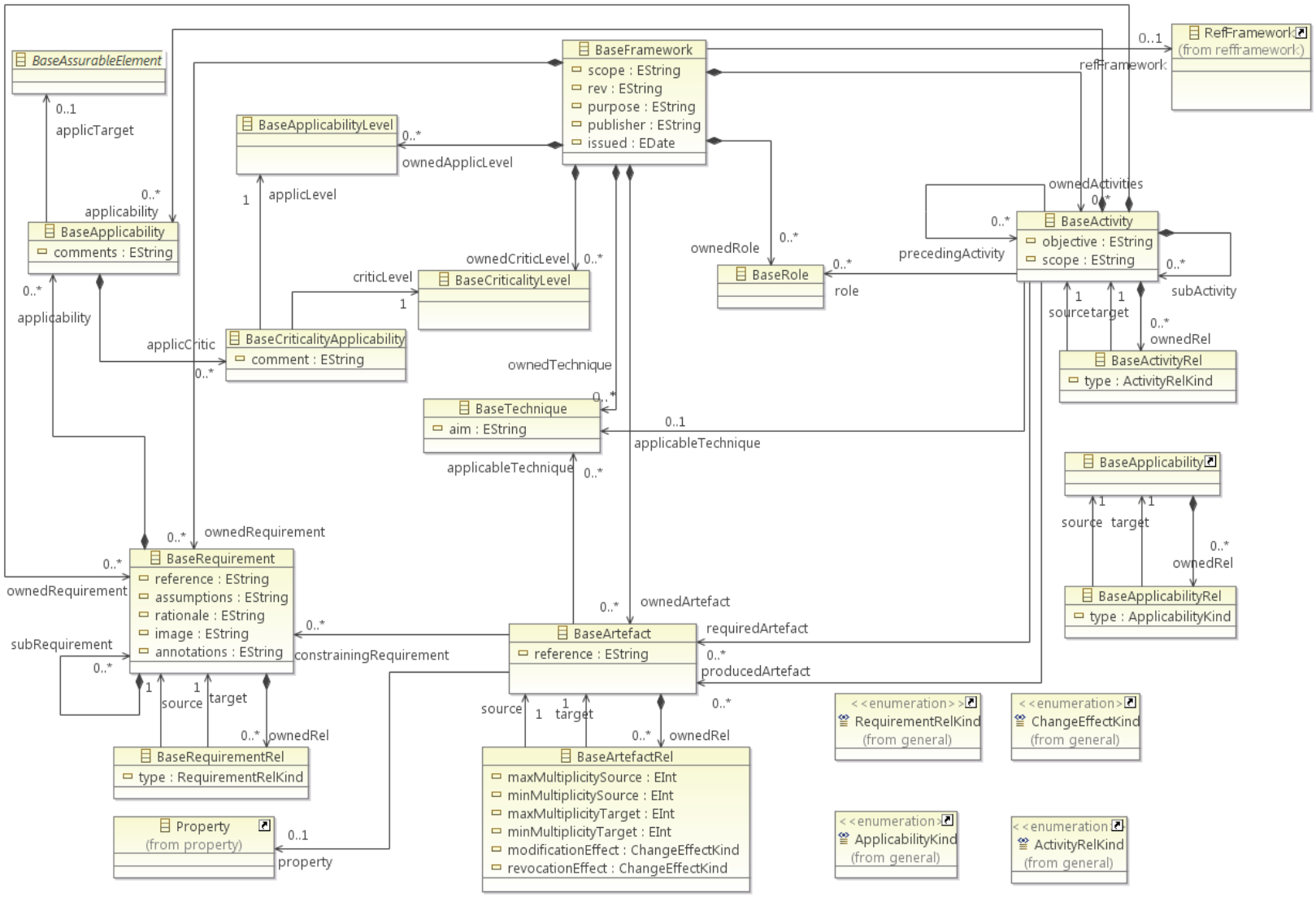


Figure 72. Baseline Definition metamodel (Part 1: Core Model Elements)

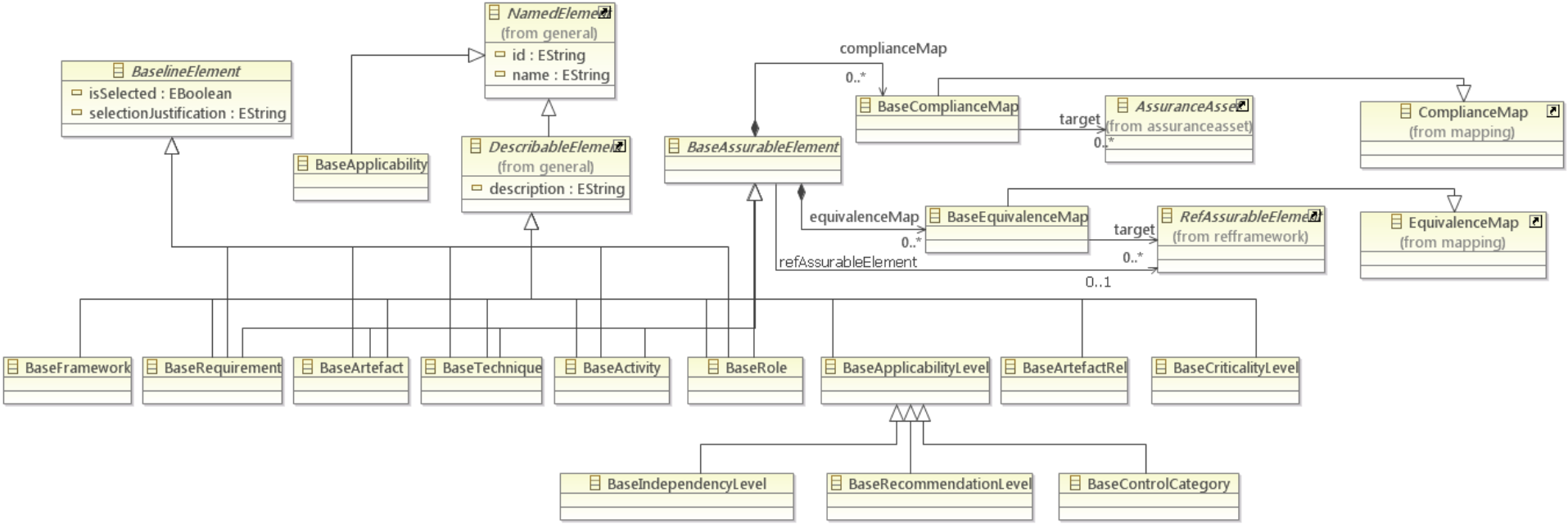


Figure 73. Baseline Definition metamodel (Part 2: Inheritance Relationship)

5.5.5.1 BaseFramework

This class corresponds to 5.5.4.1 class but it has an extra relation:

- refFramework: *RefFramework* [0..1]. The RefFramework used to create the BaseFramework.

5.5.5.2 BaseActivity

This class corresponds to 5.5.4.2.

5.5.5.3 BaseActivityRel

This class corresponds to 5.5.4.3.

5.5.5.4 BaseRole

This class corresponds to 5.5.4.4

5.5.5.5 BaseArtefact

This class corresponds to 5.5.4.5

5.5.5.6 BaseArtefactRel

This class corresponds to 5.5.4.6

5.5.5.7 BaseTechnique

This class corresponds to 5.5.4.7.

5.5.5.8 BaseRequirement

This class corresponds to 5.5.4.8.

5.5.5.9 BaseRequirementRel

This class corresponds to 5.5.4.9.

5.5.5.10 BaseCriticalityLevel

This class corresponds to 5.5.4.10.

5.5.5.11 BaseApplicabilityLevel

This class corresponds to 5.5.4.11.

5.5.5.12 BaseIndependencyLevel

This class corresponds to 5.5.4.12.

5.5.5.13 BaseRecommendationLevel

This class corresponds to 5.5.4.13.

5.5.5.14 BaseControlCategory

This class corresponds to 5.5.4.14.

5.5.5.15 BaseCriticalityApplicability

This class corresponds to 5.5.4.15.

5.5.5.16 BaseApplicability

This class corresponds to 5.5.4.16.

5.5.5.17 BaseApplicabilityRel

This class corresponds to 5.5.4.17.

5.5.5.18 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.

5.5.5.19 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.

5.5.5.20 BaseEquivalenceMap

This class corresponds to 5.5.4.19.

5.5.5.21 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.

5.5.5.22 ActivityRelKind (enumeration)

See definition in 4.1.2.3.

5.5.5.23 ChangeEffectKind (enumeration)

See definition in 4.1.2.4.

5.5.5.24 RequirementRelKind (enumeration)

See definition in 4.1.2.5.

5.5.5.25 ApplicabilityKind (enumeration)

See definition in 4.1.2.6.

5.5.5.26 Property

See definition in 4.1.3.2.

5.5.5.27 DescribableElement

See definition in 4.1.2.2.

5.5.5.28 NamedElement

See definition in 4.1.2.1.

5.5.5.29 RefAssurableElement

See definition in 5.5.4.18.

5.5.5.30 AssuranceAsset

See definition in 5.4.2.2.

5.5.5.31 ComplianceMap

See definition in 4.5.6.5.

5.5.5.32 EquivalenceMap

See definition in 4.5.6.4.

5.5.6 Implementation Vocabulary Metamodel

The implementation metamodel is a simplification of the conceptual metamodel and is described in the Figure below and the subsequent subsections.

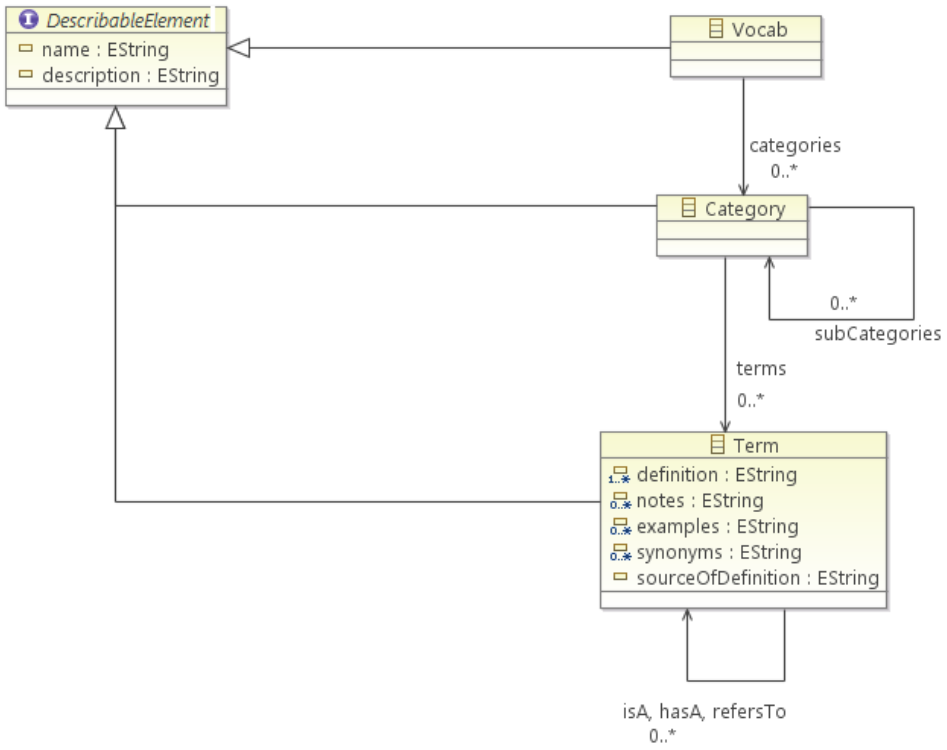


Figure 74. Vocabulary Metamodel

5.5.6.1 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)

5.5.6.2 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

5.5.6.3 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).

5.5.6.4 DescribableElement

See definition in 4.1.2.2.

5.5.7 Implementation Mapping Definition Metamodel

The implementation and the conceptual model for Mapping Definition are the same.

6. Conclusions (*)

This document describes the AMASS Reference Tool Architecture (ARTA) in its third iteration and completes the descriptions from previous iterations of this deliverable in order to ensure all the AMASS advanced functionalities are properly defined. The description includes:

- The AMASS platform characteristics.
- The functional decomposition together with the use cases.
- The high-level design of the system in term of its main components.
- The main interactions between the different components

It is helpful for the AMASS Implementation Team to use deliverable D2.4 to collaborate with other team members in implementing the architecture and to help team members understand the motivation behind architectural decisions, as the architecture evolves during the project, so that those decisions can be robustly implemented. This deliverable should be read together with design deliverables from the technical work packages (D3-6.2 and D3-6.3) in order to provide the implementations of the different use cases.

This document should also inform the AMASS team members about a high-level outline of the system and its technical motivations to whoever must maintain and evolve the architecture later (the AMASS community). It is planned so to serve as an introduction of the AMASS Platform Architecture to people from the community.

This document has been organized into three different parts:

- **Architecture Overview**. This part outlines the main AMASS architectural decisions including a vision of the overall AMASS approach, a general overview of the AMASS platform as a “system”, as well as some fundamentals and the state of the practice in industry regarding this kind of technological support.
- **Architectural Views**. This part organizes the architecture specification into a set of representative architectural viewpoints, including a use case view, structural view, and interaction view. The architecture beyond the project life is evolved.
- **CACM definition**. This part describes the Common Assurance & Certification Metamodel (CACM). The CACM aims to provide a basis for common understanding between the different domains and concerns involved in the AMASS project

The AMASS platform functionality can be summarised as follows:

- Support for system architecture specification by decomposing a system into components, including mechanisms to support compositional assurance, contract based approaches, and architectural patterns management.
- Management of argumentation information in a modular fashion, including mechanisms to support compositional safety assurance and assurance patterns management as well as to include dependency relationships.
- Support for safety and security co-analysis, and validation and verification activities by interacting in a transparent and seamless way with the different tools.
- Support for the full lifecycle of evidence artefacts and evidence chains, including evidence traceability management and impact analysis.
- Assurance project management (edition, search, transfer, etc.) of standards’ information as well as of any other information derived from them, such as interpretations about intents and mapping between standards.

- Support for the creation of assurance projects in the ARTA.
- Support for knowledge management about standards, regulations and interpretations, in a form that can be stored, retrieved, categorized, associated, searched and browsed.

Tool vendors or other stakeholders could either implement the AMASS platform following the information provided in the architectural views or use the functionalities and connect with their own external tools.

Abbreviations

AADL	Architecture Analysis & Design Language
ALF	Action Language for Foundational UML
API	Application Programming Interface
ARM	Argumentation Metamodel
ARTA	AMASS Reference Tool Architecture
ASIL	Automotive Safety Integrity Level
BMM	Business Motivation Model
BPMN	Business Process Model and Notation
CACM	Common Assurance & Certification Metamodel
CCL	Common Certification Language
CDO	Connected Data Objects
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
FMVEA	Failure Modes, Vulnerabilities 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
HIL	Human In the Loop
HMI	Human-Machine Interface
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
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
V&V	Verification & Validation
WP	Work Package

References (*)

[1] AMASS: Deliverable D2.2 AMASS reference architecture (a), November 2016

[2] AMASS: Deliverable D2.3 AMASS reference architecture (b), September 2017

[3] AMASS: Deliverable D1.5 AMASS demonstrators (b), March 2018

[4] SafeCert project https://artemis-ia.eu/project/40-nsafecer.html

[5] OPENCOSS Project http://www.opencoss-project.eu/

[6] "Certifying Boeing's airplanes," http://787updates.newairplane.com/Certification-Process, accessed: 2016-07-27.

[7] J. L. de la Vara, A. Ruiz, K. Attwood, H. Espinoza, R. K. Panesar-Walawege, A. López, I. del Río, T. Kelly. "Model-based specification of safety compliance needs for critical systems: A holistic generic metamodel", Information and Software Technology 72, 16-30

[8] 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)

[9] SafeCer Deliverable D132.2, Generic component meta-model v1.0, 2014-12-19

[10] AMASS: Deliverable D3.2 - Design of the AMASS tools and methods for architecture-driven assurance (a)

[11] AMASS: Deliverable D4.2 - Design of the AMASS tools and methods for multiconcern assurance (a)

[12] AMASS: Deliverable D5.2 - Design of the AMASS tools and methods for seamless interoperability (a)

[13] AMASS: Deliverable D6.2 - Design of the AMASS tools and methods for cross/intra-domain reuse (a)

[14] AMASS: Deliverable D3.3 - Design of the AMASS tools and methods for architecture-driven assurance (b), March 2018

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

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

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

[18] OpenCert project https://www.polarsys.org/opencert/

[19] PolarSys – Open Source Solutions for Embedded Systems https://www.polarsys.org/

[20] "ISO/IEC/IEEE 42010:2011 - Systems and software engineering - Architecture description". Iso.org. 2011-11-24. Retrieved 2013-08-06.

[21] AMASS: Deliverable D2.1 Business cases and high-level requirements. http://www.amass-ecsel.eu/sites/amass.drupal.pulsartecnalia.com/files/documents/D2.1_Business-cases-and-high-level-requirements_AMASS_final.pdf, February 2017

[22] AMASS: Deliverable D1.1 Case studies description and business impact. http://www.amass-ecsel.eu/sites/amass.drupal.pulsartecnalia.com/files/documents/D1.1_Case-studies-description-and-business-impact_AMASS_Final.pdf, November 2016

[23] CHESS Project Web: https://www.polarsys.org/chess/

[24] Origin Consulting; GSN Community Standard Version 1. 2011.

[25] SafeCer Deliverable D132.2, Generic component meta-model v1.0, 2014-12-19

[26] 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.

[27] OMG: Structured Assurance Case Metamodel (SACM). http://www.omg.org/spec/SACM/ (accessed 2016-10-31)

[28] 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

[29] E. Foundation. Eclipse process framework (epf) composer 1.0 architecture overview. http://www.eclipse.org/epf/composer architecture/. Accessed: 2016-08-29.

[30] 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)

[31] OMG: Dependability Assurance Framework for Safety-Sensitive Consumer Devices (DAF), version 1.0. http://www.omg.org/spec/DAF/ (accessed 2016-10-31)

[32] “IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems,” International Electrotechnical Commission, Geneva, Switzerland, Standard, 2010.

[33] “ISO 26262: Road vehicles Functional safety,” International Organization for Standardization, Geneva, Switzerland, Standard, 2011.

[34] “DO-178C: Software Considerations in Airborne Systems and Equipment Certification,” Radio Technical Commission for Aeronautics, Washington, USA, Standard, Jan. 2012.

[35] “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.

[36] “ARP4754A: Guidelines for Development of Civil Aircraft and Systems,” SAE International, Brussels, Belgium, Standard, 2010. http://standards.sae.org/arp4754/

[37] CHESS modelling language, https://www.polarsys.org/chess/publis/CHESSMLprofile.pdf

[38] Object Modelling Group, Semantics of Business Vocabulary and Business Rules (SBVR), 2008

[39] SafeCer Deliverable D132.2, Generic component meta-model v1.0, 2014-12-19

[40] 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.

Appendix A: Changes since the Predecessor Version D2.3 (*)

New Sections:

Section	*Title*
4	Conceptual CACM
5	Implementation CACM
Appendix A	Changes since the Predecessor Version D2.3

Modified Sections:

Section	*Title*	*Changes*
1.2	Purpose	Updated
1.4	Relation to other AMASS Tasks	Updated
2.2	Tool Architecture Outline	Updated
2.4	AMASS Tool Platform	Updated
3.1.2	Architecture-driven Assurance Use Cases	Updated to cover all the advanced AMASS functionalities related to architecture-driven assurance
3.1.3	Multi-concern Assurance Use Cases	Updated to cover all the advanced AMASS functionalities related to multi-concern assurance
3.2	Structural View	Updated
5	Conclusions	Updated
	References	Minor extensions
	Abbreviations and Definitions	Minor extensions