ECSEL Research and Innovation actions (RIA)

AMASS

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

Methodological guide for multiconcern assurance (b)
D4.8

Work Package:	WP4: Multi-Concern Assurance
Dissemination level:	PU = Public
Status:	Final
Date:	31 October 2018
Responsible partner:	Barbara Gallina (MAELARDALENS HOEGSKOLA)
Contact information:	barbara.gallina@mdh.se
Document reference:	AMASS_D4.8_WP4_MDH_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[1]

Names	Organisation
Barbara Gallina (Task Leader), Zulqarnain Haider, Shankar Iyer, Irfan Sljivo	Maelardalens Hoegskola (MDH)
Marc Sango	ALL4TEC (A4T)
Stefano Puri	Intecs (INT)
Alejandra Ruiz	Tecnalia Research & Innovation (TEC)
T.Gruber, K.Christl, S.Chlup, Ch.Schmittner	Austrian Institute of Technology (AIT)
Morayo Adedjouma, Thibaud Antignac, Bernard Botella, Huascar Espinoza	Commissariat à l'énergie atomique et aux Energies Alternatives (CEA)
Robert Bramberger, Helmut Martin, Bernhard Winkler	Virtual Vehicle Research Center (VIF)
Stefano Tonetta, Alberto Debiasi	Fondazione Bruno Kessler (FBK)

Reviewers[2]

Names	Organisation
Fredrik Warg (Peer Reviewer, D4.7)	SP Technical Research Institute of Sweden (SPS)
Garazi Juez Uriagereka (Peer Reviewer, D4.7 and D4.8)	Tecnalia Research & Innovation (TEC)
Siddhartha Verma (Peer Reviewer, D4.8)	Austrian Institute of Technology (AIT)
Cristina Martínez (Quality Manager, D4.7 and D4.8)	Tecnalia Research & Innovation (TEC)
Jose Luis de la Vara (TC Review, D4.7 and D4.8)	Universidad Carlos III de Madrid (UC3)
Stefano Puri (TC Review, D4.7)	Intecs (INT)

[1] The list includes the contributors to D4.7, which is evolved in D4.8

[2] The list includes the reviewers of D4.7, which is evolved in D4.8

TABLE OF CONTENTS

List of Figures

List of Tables

List of Codes

List of Algorithms

List of Equations

Executive Summary

This document (D4.8 Methodological guide for multi-concern assurance (b)) is the final deliverable associated with the AMASS Task 4.4 Methodological Guide for Multi-Concern Assurance, which provides information about how to use the AMASS Multiconcern Assurance approach. This is the final version and it is based on the functionality supported by the third prototype (P2) of the AMASS platform.

This deliverable is conceived as an update[3] of the previous version (D4.7 Methodological guide for multi-concern assurance (a)), which was delivered as a confidential document.

This document focuses on the techniques developed in WP4. The guide targets a diversified audience, mainly composed of process engineers, assurance engineers and development engineers.

To try to make the document self-contained, first, background information regarding the AMASS multi-concern concepts is given. Second, the AMASS multiconcern vision is recalled. Third, the potential of the tool-supported approach is illustrated via a series of workflow-diagrams. Fourth, the fundamental functionality of the tools supporting the execution of the workflows is recalled. Finally, use case-oriented scenario instantiations are used to further refine such guidelines.

To have a more general overview regarding the AMASS approach including the methods and techniques provided by other WPs, the reader is referred to D2.5 [12] as well as D3.8 [7] and D6.8 [8], which respectively provide guidance for the AMASS Architecture-driven approach and for the AMASS cross- and intra-domain reuse approach. D2.5 also includes a user manual, which contains detailed descriptions of how to use the specific functions.

[3] The sections modified with respect to D4.7 have been marked with (*), then the details about the differences and modifications are provided in Appendix A: Document changes with respect to D4.7 (*)

1. Introduction (*)

Embedded systems have significantly increased in technical complexity towards open, interconnected systems. The rise of complex Cyber-Physical Systems (CPS) has led to many initiatives to promote reuse and automation of labour-intensive activities such as the assurance of their dependability. The AMASS project builds on the results of two large-scale projects, namely OPENCOSS [26] and SafeCer [25]. These projects dealt with the assurance and certification of software-intensive critical systems using incremental and model-based approaches. Both projects focused on compositional argumentation, however, neither dealt with multiple concerns. Moreover, while the SafeCer approach was more detailed with respect to system modelling, OPENCOSS was more detailed with structuring of the assurance case. Since the two approaches are complementary, in AMASS, it has been decided to combine them and further refine them.

More specifically, SafeCer developed a generic process model given as the commonality within a configurable process line. Methodological guidelines for the EPF Composer-based Safety-oriented Process Line Engineering (SoPLE) [55] were also developed. The AMASS project consolidates and extends SoPLE to enable capturing the multi-faceted nature of assurance and thus contributing to the multi-concern assurance approach. The AMASS project also combines it with the OPENCOSS solutions for managing multi-concern compliance.

OPENCOSS elaborated solutions for assurance case structuring (i.e., vocabulary and structured expressions used in the assertions included the argumentation, as well as the composition of the arguments when they were provided by different suppliers), but the connection with system modelling was not in focus. Furthermore, the assurance case did not consider multiple concerns and how to account for their interplay. Hence, in AMASS, the compositional approach for assurance case structuring, properly connected with system modelling, and extended for multi concern assurance, has been targeted.

SafeCer also developed a generic component model and contract-based verification techniques for compositional development and certification of CPS. These have been integrated in the CHESS tool support [27]. The AMASS project consolidates and extends such support with a wider range of mono-concern focused analysis techniques for the system architecture and combines it with the OPENCOSS solutions for building an assurance case. The resulting Architecture-Driven Assurance approach (designed in D3.3 [6]) is in D4.3 [3], further extended for: multi-concerns (in particular, the interplay between safety and security is in focus); and reuse of multi-concern architectural patterns. Moreover, the approach exploits tool interoperability mechanisms (designed in D5.3 [9]) to interact with external tools for multi-concern modelling and analysis support.

Figure 1 provides a general overview of the AMASS Scientific Technical Objectives (STOs) and how they are implemented in the AMASS project by specific Work Packages (WPs). This deliverable defines the guide to be followed to apply the Multi-concern assurance approach developed in WP4. The methodological guide describes how to use the AMASS tools with help of examples and detailed process steps. The workflow is presented with the aid of activity diagrams or sequences of to-be-followed steps. The steps are meant to give an example of usage of the tool trying to cover all relevant features.

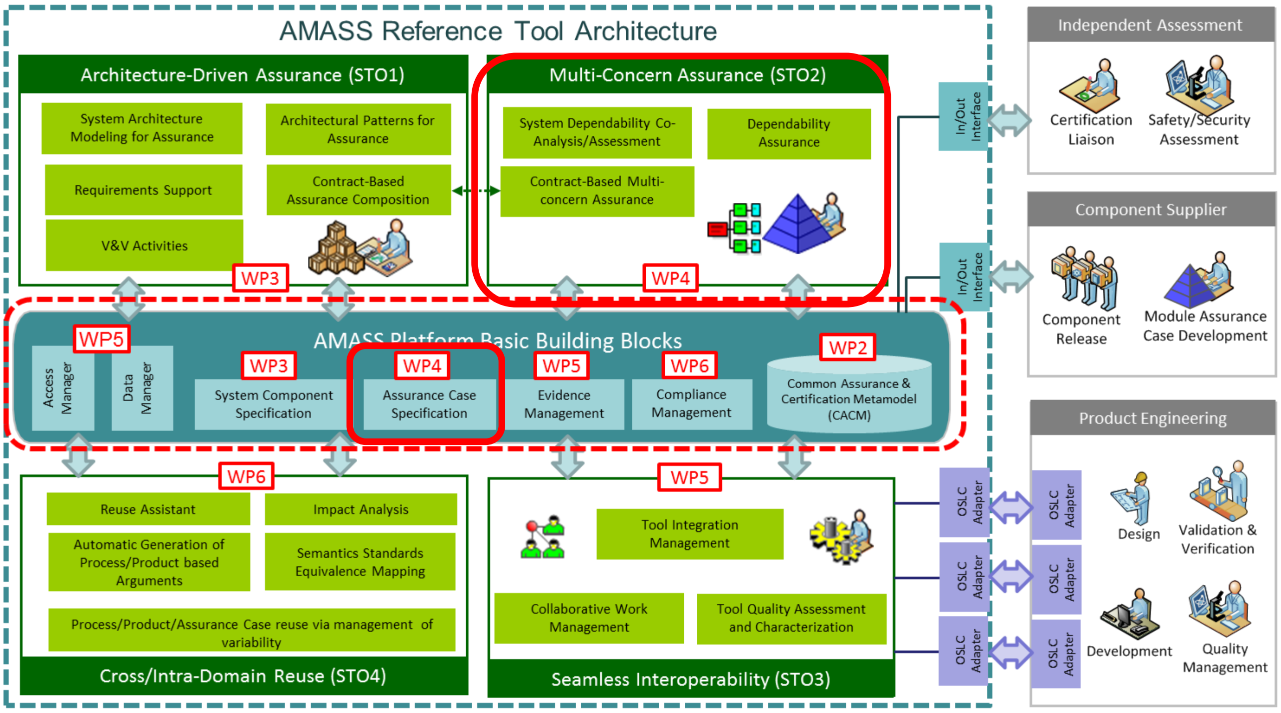

AMASS Reference Tool Architecture
Architecture-Driven Assurance (STO1)
System Architecture Modeling for Assurance
Architectural Patterns for Assurance
Requirements Support
Contract-Based Assurance Composition
V&V Activities
WP3
Multi-Concern Assurance (STO2)
System Dependability Co-Analysis/Assessment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

Figure 1. Assurance Case Specification and Multi-concern Assurance in relation to other AMASS Prototype P2 building blocks

This deliverable, first, provides an overview of the key concepts, such as contract-based multi-concern assurance, dependability assurance modelling, and system dependability co-assessment and analysis. Then, it explains what Multiconcern Assurance means, the role of the key concepts in the approach, and how the AMASS platform supports it. The core of this deliverable describes the workflows to enact Multiconcern Assurance, detailing the activities to be conducted and how to use the tool support. The workflows are presented by means of activity diagrams or sequences of steps to follow. To get a detailed explanation about the different options, the user may refer to the user manual, included in D2.5 [12]. Finally, the guide uses simple case studies to concretely describe the approach.

2. Multi-concern Assurance Overview

This chapter provides an overview of the multi-concern assurance approach. To do that, essential information is recalled: first, background information belonging to the solution space, then the vision, and, finally, the main functionalities of the individual tools composing the AMASS platform and playing an active role within WP4.

2.1. Background

The purpose of this section is to recall fundamental concepts in order to make the document self-contained and enable the understanding of the guide. The presentation of the concepts builds on top of D4.3 [3].

2.1.1. Contract Based Multi-concern Assurance (*)

The spine of an assurance case is represented by the top-level requirements and goals that should be met by the system, and the evidence supporting the confidence that those requirements are met. Typically, those top-level requirements are decomposed based on the system architecture so that assurance of the decomposed requirements supports top-level requirements to fulfil dependability properties at system level. Confidence in the requirements decomposition needs to be ensured to use the decomposed requirements also for the assurance of the top-level requirements. Assumption-guarantee contracts can assist in increasing confidence in both requirements and their decomposition.

This decomposition of requirements to ensure the system level assurance is also reflected in the system assurance case. In D4.3 [3], a proposal for the multiconcern assurance case structure was made. The system is assured for multiple concerns such that a set of system goals is developed for all the different concerns. The system goals are supported by the system requirements developed for all the different concerns. The concern-specific system goals are supported by the requirements specific to different concerns (safety, security, performance). Interplay of the concerns on all the levels where cross concern trade-off occurs (goals, requirement and components) is handled in the trade-off argument module as shown in Figure 2.

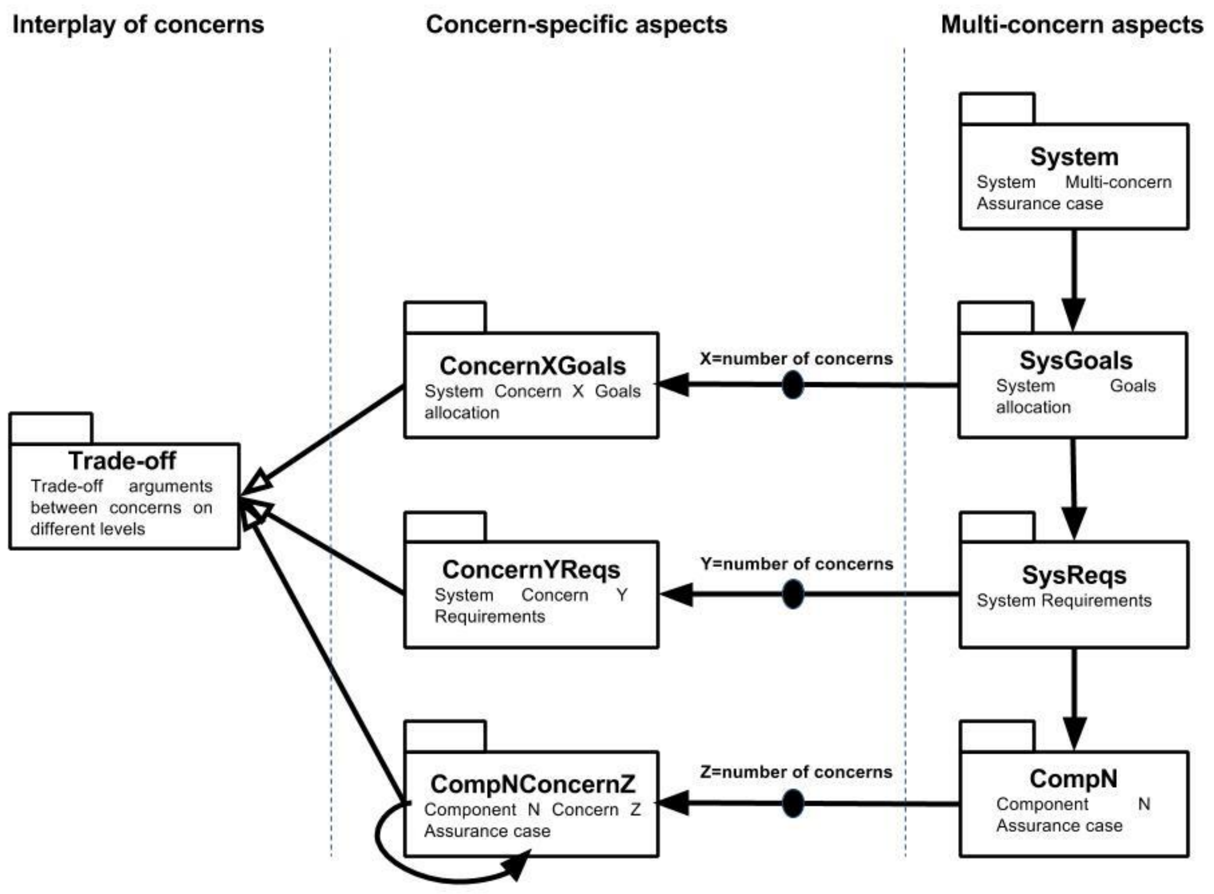


Figure 2. Multiconcern assurance case structure proposal

Considering allocation of requirements over the system architecture, contracts on the architecture elements are defined to correspond to the requirements allocated to those elements. An assumption-guarantee contract can be used to formalise a requirement such that the contract guarantees formalise the requirement by describing the behaviour of the element that implements the requirement, while the contract assumptions capture the conditions under which that behaviour is exhibited. Provided that the assumptions hold in a particular system, then the guarantee also holds, hence the corresponding requirement is met by the element in the given system. Requirement decomposition is captured by the contract refinement specification. Just as a requirement may be decomposed to a set of (sub)-requirements, the contract of an element can be refined by a set of contracts of the sub-elements.

The contract refinement analysis can be used to increase confidence in the requirements decomposition as well as to assure that a particular contract/requirement holds in the given system. To assure that a requirement is satisfied with sufficient confidence, it is necessary to argue about:

1. Is the contract or a set of contracts correctly formalising the requirement?
2. Can the inputs in the refinement analysis (i.e., can the contracts themselves be trusted? and more precisely can the corresponding element be trusted to behave according to the guarantees given the assumptions) be trusted? and
3. Can the outputs from the refinement analysis (i.e., can we the tool itself be trusted) be trusted?

Assuring these aspects allows the outputs from the contract refinement analysis to be used to support both requirements decomposition and requirement satisfaction. The first point may be addressed for example by inspection of the requirement and the corresponding contract guarantees, while testing or simulation can be used to support the second aspect. The third aspect may be addressed by verification of the tool and methodology used for contract checking. The last aspect is related to the tool qualification activities and the level of confidence put on it.

Considering that each requirement may be related to one or more different concerns such as safety and security, assurance of different contracts supports assurance of those concerns related to that contract. Furthermore, as the contracts connect additional information to the requirement in terms of assumptions, the contract-based assurance supports identification of interactions of those formalisable requirements across concerns. Dependency, conflicting as well as supporting relationships between elements and their concern-specific requirements can be highlighted through contract-based assurance.

2.1.1.1. Contract-based Trade-off Analysis in Parameterized Architectures

Parametrized architectures, as defined and developed in WP3, provide the means to analyse the system architecture in different configurations. Each configuration may enable/disable some components, ports, connections, and contracts. Different configurations can be analysed and compared with respect to different aspects: contract refinement, satisfaction of formal properties, fault tolerance, minimal cut sets, reliability measures. Such an approach was for example followed in the analysis of different configuration of the next generation of air traffic control design [75].

Comparing the different configurations allows the designer to perform *trade-off analysis* and *design space exploration*. Architectural choices are supported by the mentioned analysis results. In particular, the choice whether adding or removing a function (represented by a block or by a contract), enabling or disabling a redundancy, or other similar changes is supported by checking which functional and non-functional properties hold in the different configurations. This trade-off analysis is enhanced by the information about the concern addressed by the different properties and contracts: the analysis provides a direct way to evaluate the impact of the trading-off architectural elements on the multiconcern represented by properties and contracts.

2.1.2. Dependability Assurance Case Modelling

As it was recalled in D4.1 [2], originally, when the necessity of demonstrating safety management emerged [58], the concept of *safety case* was introduced. Decade after decade, this concept has evolved to include other properties such as security, performance, conformance, trust, etc. Nowadays, the concept of *Assurance Case* is used to refer to a case that covers any critical property to be assured.

An Assurance Case is a set of auditable claims, arguments, and evidences created to support the claim that a defined system/service will satisfy some particular requirements [57]. Assurance cases use a structured set of arguments and a corresponding body of evidence to justify that a system satisfies specific claims with respect to its properties (i.e. safety, security, reliability, availability, etc.).

With Dependability Assurance Case modelling, advantages of two main concepts are taken. On the one hand the compositional argumentation and, on the other hand, the power of argumentation applied on dependability.

Compositional argumentation means to deal with the challenge of complexity and length of the assurance cases. By adopting a modular, compositional, approach to the assurance case construction it may be possible to:

- Justifiably limit the extent of the assurance case modification and revalidation required following anticipated system changes.
- Support (and justify) extensions and modifications to a ‘baseline’ assurance case.
- Establish a family of assurance case variants to justify the dependability of a system in different configurations.

This approach establishes a modular and compositional construction for assurance cases that has a correspondence with modular structure of the underlying architecture. As with system architecture, the assurance engineer should establish interfaces between the modular elements of the assurance (safety, security, conformance...) justification such that the assurance case elements may be adequately composed, removed and replaced. Similarly, it will be necessary to establish the assurance argument infrastructure required in order to support modular reasoning.

In order to provide assurance of the system to carry out its intended function in its intended context, the relationships between the dependability aspects of the system (safety, security availability…), the decisions made during the development of the system to accommodate them, and the effects of these decisions and any other concerns which they impact (in this case, maintainability, performance, and potentially security) have to be recorded.

Assurance cases are not a fixed document but rather a living document, as Denney, Pai and Habli proposed in [73], "Dynamic Safety Cases" should be targeted. Artefacts should be checked, validated and updated based on actual feedback data. With this conception of dynamic assurance case, in AMASS, the need for an explicit notation that shows that a claim has an impact (to reassure, to dismiss or no impact) in another claim has been identified. More specifically, the following relationships between dependability properties in the assurance case have been identified:

- *Dependency relationship*. The claim A of one attribute depends on the fulfilment of claim B of another attribute. For example, a fail-safe claim of attribute safety depends on the claim that the safety instrumentation system is not tampered of attribute security.
- *Conflicting relationship*. The assurance measure of attribute A is in conflict with the assurance measure of attribute B. For example, a strong password or blocking a terminal after several failed login attempts for security conflicts with the emergency shutdown for safety. Resolution of such a conflict need to be noted in the Assurance Case.
- *Supporting relationship*. The assurance measure of attribute A is also applicable to assurance of attribute B, such that one assurance measure can be used to replace two separate ones if the attributes are considered and addressed individually. For example, encryption can be used for both: for confidentiality in terms of security and to check data integrity regarding safety. This means two goals can be addressed by one argumentation.

Another challenge that security experts need to face is the temporary effectivity of the assurance decisions. As security threats evolve in time, as attacks improve, the security mechanisms put in place need to be re-assured after some time. Assurance cases need to be checked periodically to ensure that evidence used to support the safety and security properties is still valid [60] and if not, provide an impact analysis and modify the system to ensure that the vulnerabilities are mitigated and/or avoided. Assurance cases should not be seen as a static tool but rather as a dynamic and living mechanism that supports safety and security responsible during the impact analysis task.

2.1.3. Process-related Dependability Co-assessment

To achieve a fully functional automated car, car manufacturers are constantly increasing the complexity of the functions. Developers of these vehicles have to deal with functional safety on the one hand and cybersecurity on the other hand. In that context, cybersecurity gets more and more important because automated driving needs information transfer from outside of the vehicle, e.g. between vehicle and environment (keyword "V2X – communication").

This subsubsection presents the concept of co-engineering and how it could be implemented via Security-informed Safety-oriented Process Line Engineering (SiSoPLE) [55], supported by the integration of EPF Composer (shortened EPF-C) [18] and BVR Tool [24]. Co-engineering supports the combination of cross concern activities to a joint process. This method is used during process development (see Figure 3) and supports Process-related Dependability Co-assessment. Different domains like automotive and avionics have different requirements, which lead to different processes and workflows. From another perspective, processes often deal with similar concerns like functional safety, cybersecurity and other quality-related concerns. This point of view makes clear that many methods are useable in different realisations in various domains. Product developers follow well-defined domain specific processes and workflows, which should cover a wide spectrum of concerns.

The interaction between functional safety and cybersecurity methodologies has to be defined systematically. A “Safety-Security-Co-Engineering” approach has to be offered. The activities concerning this approach belong to the block “Process development” in the “Process framework overview” in Figure 3. The approach compares relevant standards, for example ISO 26262 for functional safety in the automotive domain and SAE J3061 for cybersecurity in vehicle systems and identifies commonalities and variabilities of those standards.

Note: The successor to SAE J3061 is under joint development between ISO and SAE, which is called ISO/SAE 21434 - Road Vehicles - Cybersecurity Engineering.

After identification of relevant standards, the framework leads via process development to process management. Additional compliance management and argumentation management is considered. The following subsection regards only co-engineering which is part of process development.

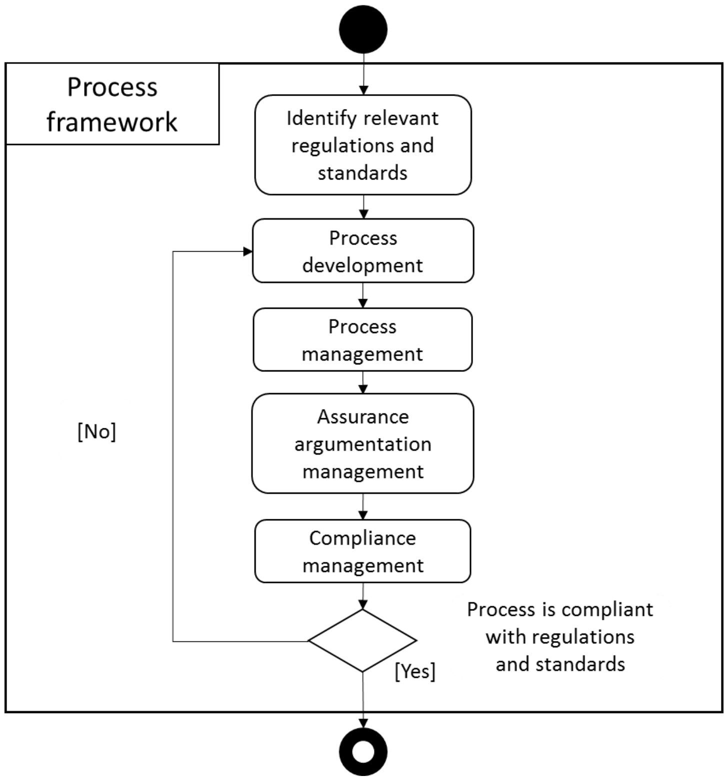

Process framework
Identify relevant regulations and standards
Process development
Process management
Assurance argumentation management
Compliance management
[No]
Process is compliant with regulations and standards
[Yes]

Figure 3. Process framework overview

Standards allow flexible but thoroughly justified interpretations and customisations, which can be modelled as variabilities. Differences between project specific processes, which arise through instantiation of identical base processes may be interpreted as variabilities. Variable activities can be managed with the methodology shown in Section 3.3.2 - BVR Workflow. To deal with commonalities based on a co-engineering approach, we must define two types of commonality. The first definition is related to the Safety-oriented Process Line (SoPL) [30], which deals with single concern – cross domain processes. In this case, common activities are identified in different domains (e.g. functional safety in the automotive and industrial domain).

For cross concern topics, we have to extend the primary definition of single concern commonality. The intention is to “maximize” co-engineering activities and deal with variability in a way that makes elaborated processes reusable. Activities in cross concern applications, which must be executed in any case, are called safety security co-engineering activities instead of single concern “commonality”. The main difference is that co-engineering activities do not necessarily contain common activities, but they lead to a common goal. We must make sure that co-engineering guarantees interaction between different concerns, in our example safety and security related activities. This interaction guarantees functional safety at the demanded level, and it makes sure that cyber-security issues are considered (in our example based on ISO 26262 and SAE J3061). SAE J3061 risk levels quantify the risk of successful cyberattacks. Risk levels are derived based on “attack potential”, “attack probability”, “severity” and “controllability”. In our case it is a criterion that indicates the risk that functional safety can possibly be levered out by an attacker in certain circumstances. The task is to combine two different concerns, which apparently may be considered independently, but they are not. In our framework, activities concerning functional safety and cybersecurity are considered in joint activities. In the concept phase, ISO 26262 demands that the activity Hazard Analysis and Risk Assessment (HARA) must be performed. A process, which beyond safety also considers security, has also to perform Threat Analysis and Risk Assessment (TARA). That process must consider the potential dependence between HARA and TARA and has to perform these two activities in parallel but intertwined.

Safety engineer and security engineer are different roles performed by different persons and depending on the role the safety or security activities will be executed. However, in this approach both roles need to be synchronized and exchanging information between teams. One of the activities that should be executed in combination is analysis approaches like System-theoretic Process Analysis for Security (STPA-Sec) [31] for concept phase and Failure Mode, Vulnerabilities and Effects Analysis (FMVEA) [32] for system level are able to identify interdependence between functional safety and cybersecurity. Identification of hazards and potential causes is an indispensable presupposition for a safe and secure system. We must identify hazards and threats from both areas because insufficient controls can lead to unsafe control actions, independent whether the cause is related to a hardware fault (classic safety-oriented view) or to a security issue. In some cases, we will identify cybersecurity risks, which influence only non-safety areas (e.g. privacy) but they are out of scope from our safety perspective. Section 2.1.4 provides additional information concerning co-analysis methods.

The interest is to define measures, which are appropriate to mitigate any identified risks. The co-engineering approach must cover hazards, which arise due to the combination of safety and security risks. As a consequence, we need to perform a safety and security co-analysis, which should guarantee that we identify any additional potential hazards, which would stay undiscovered if only one discipline is examined in an isolated way. To make sure that measures from competitive disciplines do not influence each other in a non-admissible way, we have to consider a trade-off in the risk reduction measures. In other words, developers have to decide how much impact is allowed for each single safety and security measure. A metric has to be developed as an aid to find out the balance and as an argument why a specific safety-security constellation has been chosen. Finally, all arguments have to be collected in the assurance case, which covers the integrated and harmonized safety and security case. In an assessment, which deals with safety and security, evidence is needed to argue why the trade-off between safety and security conforms with standards from both domains.

The tool EPF-C is used to model the safety and security co-engineering process and the tool WEFACT is used to execute the process workflow and gather all the required evidences for the argumentation. An example which shows how the two tools are used can be found in D4.3 [3].

2.1.4. System Dependability Co-Analysis

Co-analysis covers a wide range of methods and techniques to identify safety hazards and security threats, which are often the activities in the early stage of a product/system development lifecycle, e.g. in the requirements engineering as well as the design phase. These analyses are also regarded as approaches to risk assessment, because the goal of the analyses is often to identify safety and security risks.

In the context of the AMASS project, more precisely in the context of D4.3 [3], the following methods were identified as an initial reference for co-analysis:

- The SAHARA method, which combines the automotive hazard analysis and risk assessment (HARA) with the security domain STRIDE approach to quantify impacts of security threats and safety hazards on system concepts at initial concept phase.
- The FMVEA Method, which was developed in the context of the ARROWHEAD project [59] and extends the established Failure Mode and Effect Analysis with security related threat modes.

These two methods are expected to be further developed during the third iteration of the AMASS prototype.

Besides these methods, additional two methods will strengthen the AMASS Co-Analysis approach:

- The joint analysis performed via fault trees and attack trees conducted via Safety Architect [14], as well as the security analysis performed via the EBIOS (Expression des Besoins et Identification des Objectifs de Sécurité - Expression of Needs and Identification of Security Objectives) method conducted via Cyber Architect [15] . The results of these analyses are expected to be exchanged with the AMASS platform.
- Failure Logic Analysis via ConcertoFLA [34], which is a result of the EU ARTEMIS CONCERTO project [88] and was extensively recalled in D4.5 [4] as well as in D4.6 [11].

2.2. Vision

The core vision of the AMASS Multiconcern assurance consists of the exploitation of:

(1) Synergies between safety and security (among other dependability properties), as it was discussed in [55]. Such synergies offer clear opportunities for co-assessment and co-analysis. In AMASS, co-assessment is enabled via the integration of an open source process engineering tool and a variability management tool, plus explicitly indicate equivalences between activities, artefacts and requirements in the standards. Co-analysis is enabled via a combination of open-source and non-open-source analysis techniques, which are expected to offer different advantages and trade-off capabilities and evidence.

(2) Contract-based approaches for compositional assurance developed in OPENCOSS and SafeCer. These approaches, which were extended in D4.3 [3] and partially implemented in D4.6 [11], include a multi-concern perspective enabling: the decomposition of the requirements (related to different concerns) onto the architecture components; the semi-automatic derivation of analysis results from the architecture; the definition of a safety/security/multi-concern concept with mitigation mechanisms on top of the architecture.

2.3. Tool Support Overview

The tool support is based on a collection of Eclipse plugins that provide the different functionalities necessary to perform the Multiconcern Assurance Approach. In particular, it includes: **EPF Composer plugins** to model the processes representing e.g., safety and or security plans; **Papyrus plugins** to model SysML diagrams; **CHESS plugins** to design and perform different model-based analyses, and **OpenCert plugins** to create and link assurance argument fragments. These plugins are part of the AMASS platform, which provides the user a single user interface hiding the complexity of the underlying tool architecture. The AMASS platform interacts with external backend tools to provide analysis results (via Safety Architect, Papyrus for Safety and Security Engineering, and FMVEA) or to execute the process plans (WEFACT).

Except for FMVEA and WEFACT, the following subsections recall only essential information regarding the main functionalities implemented within the different tools. A more extensive description of the tools was given in D4.3 [3]. Concerning FMVEA and WEFACT, instead, since a new version of these tools is in the process to be released, a more detailed information is provided to enable the reader to have a more concrete idea of the potential of the coming support.

2.3.1. CHESS

CHESS Eclipse Polarsys project [36][27] provides support for system and software modelling, analysis and implementation. The CHESS modelling language (CHESSML)[4] is implemented as a profile of UML, SysML and MARTE modelling languages. CHESSML supports component, contract-based design and the modelling of timing and dependability concerns. Analysis support is made available by using the information provided within the model and by providing seamless integration with tools for dependability analysis, like ConcertoFLA for failure propagation (see 2.3.1.1) and multi-concern fault tree analysis (see 3.5.3), xSAP/OCRA for fault tree analysis, contract-based analysis, like OCRA, and timing analysis, like MAST[5]. Regarding software, the specific CHESS methodology [36] for software modelling, analysis and implementation is supported, by offering a model driven approach with code generation facility (currently Ada is supported as target language).

2.3.1.1. ConcertoFLA

The AMASS platform, via inclusion of CHESS toolset, also includes the plugin which implements ConcertoFLA, a technique for qualitative dependability analysis. More specifically, this plugin retrieves the dependability-related information (behaviour of the components in the presence of faults) and exploits it to calculate the behaviour at system level. The analysis results are then back-propagated and annotated on the original model.

2.3.2. OpenCert –Assurance Case Editor

This feature manages argumentation information in a modular fashion. 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.

During the safety argumentation phase the assurance case editor is used to define an argumentation model using the GSN graphical notation [5]. Argumentation deals with (a) direct technical arguments of safety, required behaviour from components, (b) compliance arguments about how prevailing standard has been sufficiently addressed, and (c) backing confidence arguments about adequacy of arguments and evidence presented (e.g. sufficiency of Hazard and Risk Assessment).

It also includes mechanisms to support assurance patterns management which offer the possibility to take advantage of reusing best practices. The argumentation editor is able to re-use predefined patterns just by “drag and drop” the pattern into the working area. Similarly, previously created argument modules can be included in the actual diagram just by “drag and drop”.

2.3.3. FMVEA (*)

A new browser-based FMVEA tool has been developed recently (spring/summer 2018) and is available in the third iteration of the AMASS platform (P2).

FMVEA extends the well-introduced FMEA by security aspects and can be used in those phases of the lifecycle where a semi-quantitative FMEA is applicable. This applies first to the concept phase where the traditional safety-oriented HARA (Hazard Analysis and Risk Assessment) can be enhanced by the assessment of security risks (TARA – Threat Analysis and Risk Assessment) when FMVEA is used. Further, FMVEA is beneficial in later development phases when an architectural or a design choice has been taken, or a concrete implementation is in place, and the resulting system is to be analysed in more detail with respect to safety and security risks. The goal can be to verify that the designed or implemented safety functions and security controls satisfy the previously stated safety and security requirements, or to detect

[4] https://www.polarsys.org/chess/publis/CHESSMLprofile.pdf

[5] https://mast.unican.es/

additional risks resulting from the concrete design or implementation that have not yet been identified in the early HARA/TARA phase.

FMVEA – Failure Modes, Vulnerabilities and Effects Analysis is a method developed since 2014 for supporting a combined safety and security analysis. The method tries to cope with the problem that the risk of safety threats can be calculated as a quantitative value based on the stochastic failure probability, but there is no comparable numeric value that can be given for security hazards because many existing vulnerabilities are yet unknown and there is no analytic method available to determine the attack probabilities – criminality is not really predictable. FMVEA therefore adds a traditional semi-quantitative security assessment approach, namely Microsoft's STRIDE classification scheme, to the classical safety-oriented method FMEA (Failure Modes and Effects Analysis). STRIDE considers the following security threat mechanisms (whose initials form the acronym STRIDE):

- **S**poofing of user identity
- **T**ampering
- **R**epudiation
- **I**nformation disclosure (privacy breach or data leak)
- **D**enial of service (D.o.S)
- **E**levation of privilege

Figure 4 shows the FMEA process (white) extended by the security-related aspects (green).

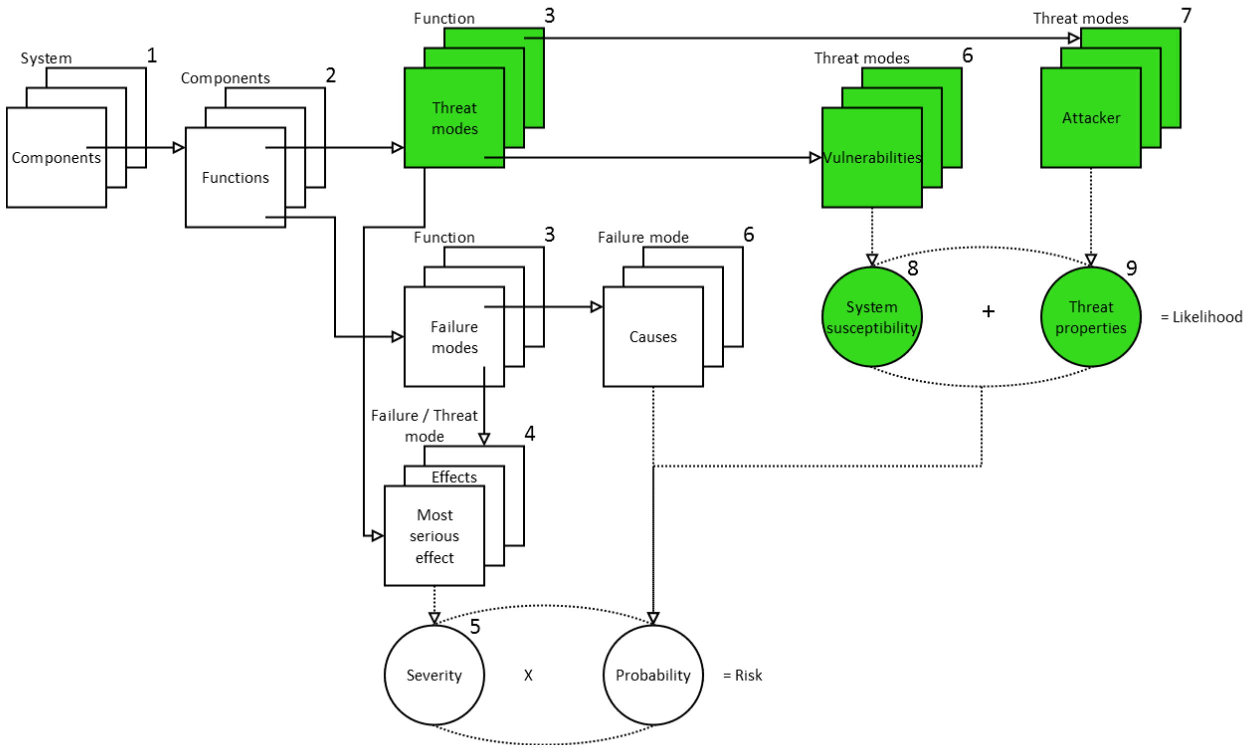


Figure 4. Security-oriented FMVEA elements complementing FMEA

For each Threat Mode, experts assess System Susceptibility and Threat Properties by estimating semi quantitative values for related attributes:

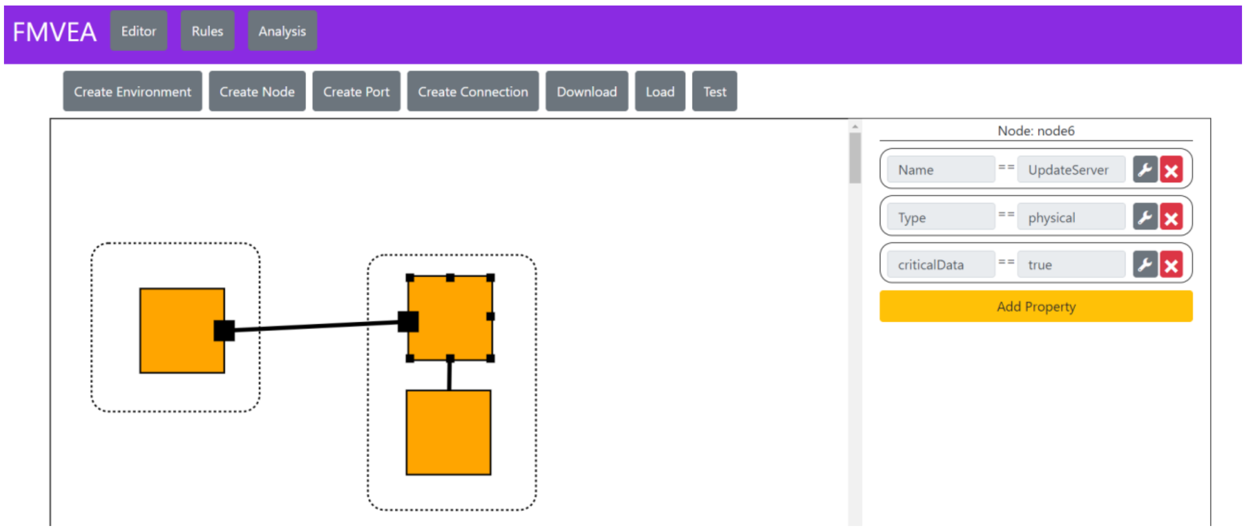
- System Susceptibility is the sum of:
  - Reachability (1 = no network, 2 = private network, 3 = public network)
  - Unusualness (1 = restricted, 2 = commercially available, 3 = standard)
- Threat Properties is the sum of:
  - Motivation (1 = opportunity target, 2 = mildly interested, 3 = main target)
  - Capabilities (1 = low, 2 = medium, 3 = high)

- Attack Likelihood is the sum of System Susceptibility and Threat Properties; this yields values between 4 and 12 and is a semi-quantitative indicator for the attack likelihood.

The FMVEA tool realizes a partly automated implementation of the FMVEA method [70]. Basically, FMVEA takes the FMEA approach and complements it with security by analysing, in addition, threats and vulnerabilities of the item under consideration.

The FMVEA tool interfaces with the AMASS platform on the one hand with the SysML model provided e.g. with Papyrus, and on the other hand with the created safety and security requirements via ReqIF format, which can be imported in the AMASS platform. More details about the integration and the interfacing platform can be found in D4.6 [11].

Figure 5 shows the FMVEA model editor user interface.



Node: node6

Name	==	UpdateServer
Type	==	physical
criticalData	==	true

Figure 5. User Interface of the FMVEA model editor.

It is possible to edit the model within the FMVEA tool or, alternatively, to reuse a model from the AMASS platform created e.g. with Papyrus, and enhance it with the respective dependability properties in the FMVEA tool. After the model instances of the system including these properties are ready, they are analysed with respect to safety and security and saved again in this scheme.

Efficient security analysis can be obtained using a pre-populated threats database, which allows semi-automatic security analysis. Similarly, a semi-automatic safety analysis is supported when a predefined failure database is used. Irrespective of whether automatic or manual analyses have been chosen, FMVEA allows extending the model according to the resulting combined set of safety and security requirements and storing it – via the SysML interface – in the AMASS platform instance.

2.3.4. EPF Composer and BVR Tool

The Eclipse Process Framework (EPF) Composer [23] is an integrated development environment which is built on top of the Eclipse platform and works as a stand-alone application. The EPF Composer provides a process-management platform based on SPEM [19] for authoring, maintaining and sharing development process frameworks between the various stake-holders of the software development organization. The outcomes of processes, which are represented in the EPF Composer as work products, provide evidence supporting process and product argumentation. This provides a means for co-engineering of safety and cybersecurity analysis, development and argumentation.

As it was recalled in D6.3 [10], BVR (Base Variability Resolution) [61] is a language built on top of CVL (Common Variability Language) [62] to enable variability modelling in the context of the engineering of families of safety-critical systems. BVR is a result of the VARIES project [64]. The specification of the BVR meta-model is given in VARIES D4.2 [63].

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

- VSpec models are an evolution of the Feature-Oriented Domain Analysis (FODA) [65]. More specifically, VSpec extends FODA by including additional concepts such as variables, references and multiplicities. Constraints by using the Basic Constraint Language (BCL) can also be added to specify cross-cutting constraints that constrain inclusion/exclusion within a subtree based on choices on other subtrees. The grammar of BCL is given in Appendix of D6.3 [10].
- Resolution models, which specify the desired inclusion/exclusion choices for the specific configuration/resolution. Note that to confirm whether the resolution corresponds to the VSpec model, a validation process might be executed. The Software Product Line Covering Array (SPLCA) tool is integrated with the BVR bundle for checking constraints and structural consistency of the resolution [66].

Realization models, which specify the *placements*[6] and *replacements* within the *fragment substitutions*. A Fragment substitution is an operation that, if executed, substitutes a model fragment (placement fragment) for another (replacement fragment).

The process model developed using the EPF Composer serves as the Base Model to the BVR Tool, which is used to model variability and derive specific processes based on feature constraints and cardinality.

2.3.5. WEFACT

The goal of the workflow engine WEFACT is to support the entire engineering lifecycle of safety and or security relevant systems based on pre-defined processes. To achieve this goal every project in WEFACT contains Requirements, Processes and Workflow Tools.

WEFACT is an (independent) Eclipse RCP application, which operates on a PostgreSQL database. As WEFACT is an external tool, this database is independent of the AMASS platform database.

WEFACT provides the following main features:

- selecting a project or creating a new one
- defining users and roles
- importing requirements (currently from a DOORS database, for the future, also ReqIF import is planned) or defining them in WEFACT
- defining activities to be performed by the workflow engine
- assigning activities to requirements and to tools (including parameters as well as input and output directories), thus supporting traceability
- executing these activities (by invoking the tools)
- setting the fulfilment status of the requirements to PASS or FAIL, depending on the result of the activities.

These basic features are complemented by the following functionalities:

- Definition of user accounts and user authorization.
- Importing UMA process models created in EPF-C. The imported activities form then the basis for the V&V activities in WEFACT.
- Assigning tools. A list of tools is maintained in WEFACT and individually assigned to V&V activities.

[6] A placement fragment is a set of elements forming a conceptual hole in a base model, which may be replaced by a replacement fragment [67].

- Traceability.

WEFACT is an Eclipse application, not an Eclipse plugin; thus, no Eclipse installation is required but WEFACT is started as an independent executable. In order to start working with WEFACT, the user first has to register with his credentials (see Figure 6).

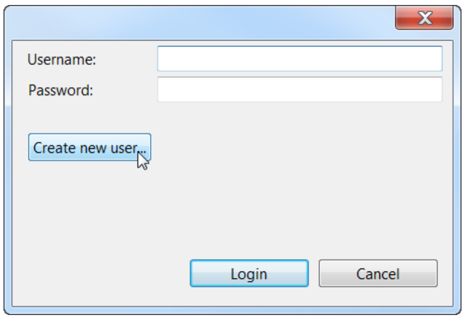


Figure 6. WEFACT user authorisation

and to select an existing project or create a new one (see Figure 7).

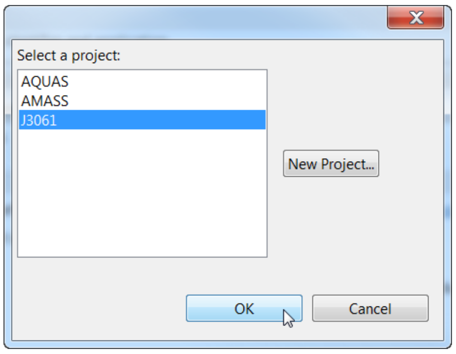
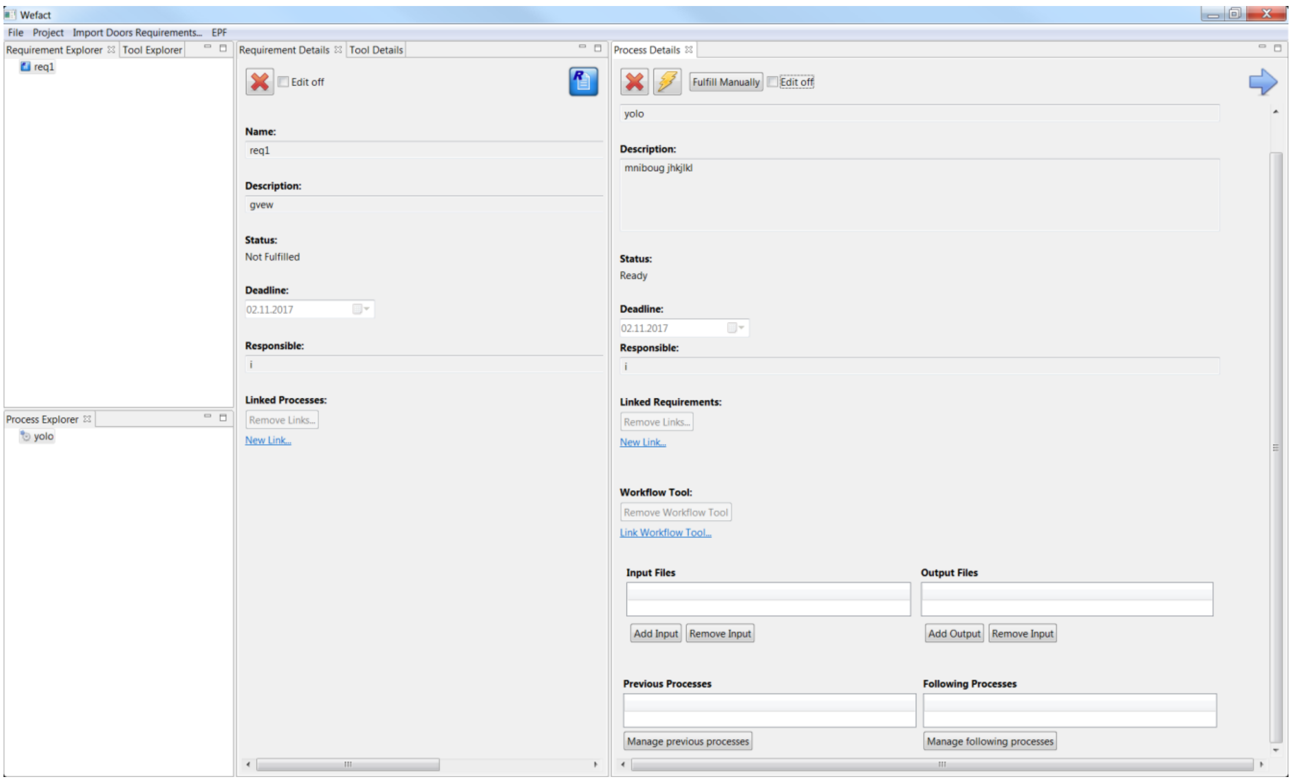


Figure 7. WEFACT project selection dialog box

Then the project is displayed in the main user interface of WEFACT, as shown in Figure 8.



Name: req1
Description: gvew
Status: Not Fulfilled
Deadline: 02.11.2017
Responsible: i

yolo
Description: mniboug jhkjlkl
Status: Ready
Deadline: 02.11.2017
Responsible: i

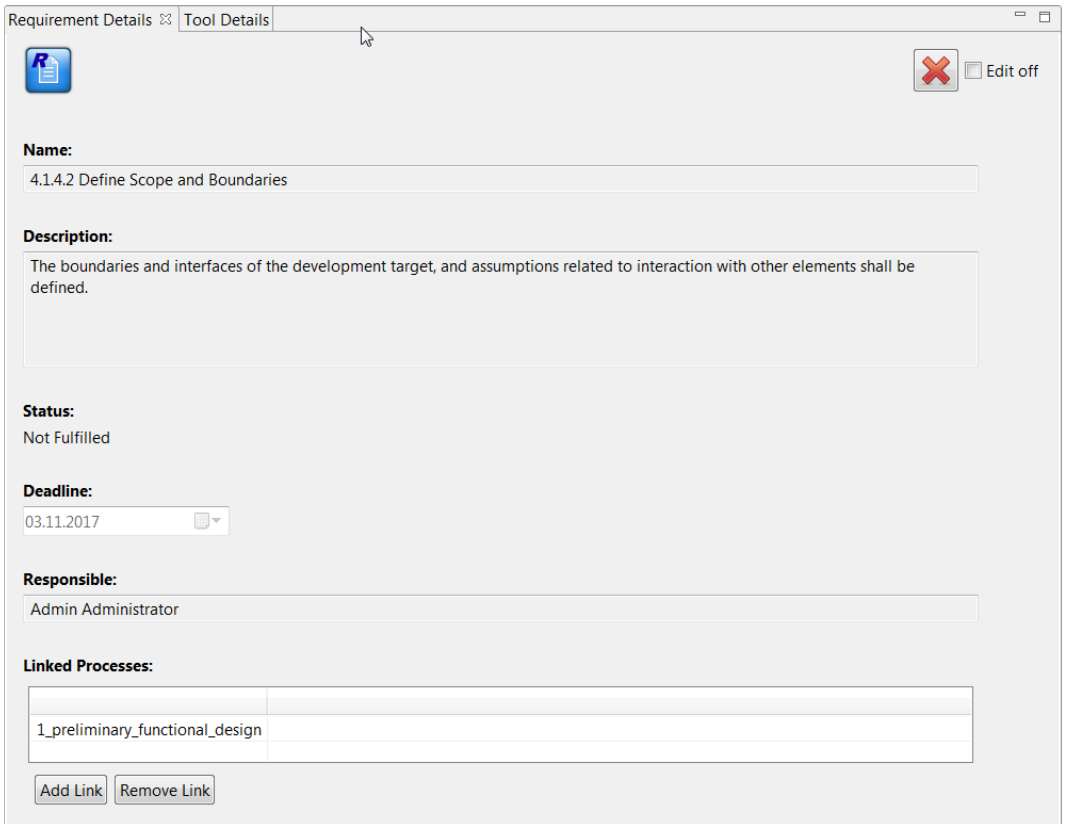
Figure 8. WEFACT user interface

The default WEFACT GUI is divided into three main parts. The usual process flow inside the application is from the left-hand side to the right-hand side. On the left-hand side, there are 3 different explorers. This area displays the project specific requirements, processes and tools and their structure. The details of the selected requirement can be viewed and edited in the part on the right side of the explorers called "Requirement Details".

Details on how the user interface is operated can be found in the WEFACT user manual [37]. In the following sections, terms are explained and guidance is given how WEFACT shall be applied, in particular in the context of AMASS assurance projects.

Requirements

As mentioned above, WEFACT is a requirements-based workflow engine. The tool allows to create and delete requirements but also to import them from external sources (currently DOORS databases). Moreover, they can be locked against unintended modification by ticking the respective checkbox. Figure 9 shows the input-box for the requirements in WEFACT.



Name:
4.1.4.2 Define Scope and Boundaries

Description:
The boundaries and interfaces of the development target, and assumptions related to interaction with other elements shall be defined.

Status:
Not Fulfilled

Deadline:
03.11.2017

Responsible:
Admin Administrator

Linked Processes:
1_preliminary_functional_design

Figure 9. Requirement data input in WEFACT

Requirements are defined as the entities needed to achieve the objectives of the project. This includes process and product requirements. Requirements can be structured in different levels, where a top-level Requirement can be seen as the sum of its sublevel Requirements. Once all sublevel Requirements are fulfilled, the top-level Requirement enters the state of completion. A Requirement can hold a connection to predefined processes (V&V activities). If all processes are executed successfully, the Requirement's status changes to "fulfilled".

Requirements have a responsible user assigned and can come from different sources. In a typical assurance workflow, process requirements are modelled in EPF-C and imported in WEFACT. Product requirements, in turn, are often created using tools, sometimes they are simple Excel files. WEFACT allows also the import of DOORS requirements, and for a future version also ReqIF import is planned.

Processes/Activities

WEFACT allows to assign processes (activities) to a requirement which shall show its validity. In the user interface, the Section "Linked Processes" shows requirements that need to be fulfilled and that are linked to this process. By selecting "Add Link…", a process can be assigned to a requirement. By clicking "Remove Link…" certain links can be removed.

Such an activity usually includes a call to a tool ("Workflow tool"), and a due deadline can be defined for processing it. For the selected tool, input artefacts ("Input Files") and output artefacts ("Output Files") shall be defined. A button allows then to start the process, which yields as a result whether PASS or FAIL, and successful activities (PASS) lead to changing the status of the requirement to "fulfilled".

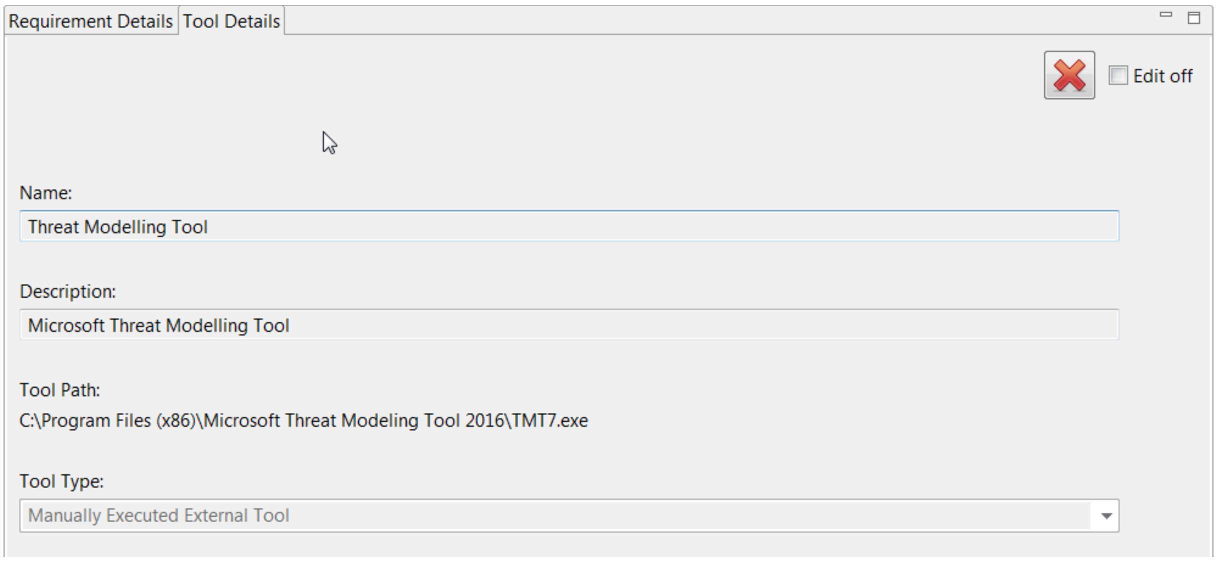
If required, subsequent calls of tools in a defined and success-dependent sequence can be forced by defining activities per tool and linking them in the desired sequence by defining "Previous Processes" and "Following Processes". In this case, the process can only be executed when all predecessor processes have been executed successfully. This can, for instance, be used to start an automatic test case generation tool before running the test created cases.

Apart from tool-based requirement verification, WEFACT allows also user decisions as basis for setting a process result – without running the activity. To enable this, a “Fulfil Manually” button has to be ticked.

Similar as requirements, also processes can be secured against unintended modification by ticking a button, and also processes have a status.

Tools

Figure 10 shows the dialog box for defining workflow tools.



Name:
Threat Modelling Tool

Description:
Microsoft Threat Modelling Tool

Tool Path:
C:\Program Files (x86)\Microsoft Threat Modeling Tool 2016\TMT7.exe

Tool Type:
Manually Executed External Tool

Figure 10. Tool definition box in WEFACT

As mentioned earlier, WEFACT supports assigning a tool to a process. This is done by writing the URL of the executable or script file into the text field “Tool path”. WEFACT supports different types of tools w.r.t. the call mechanism, namely manual/automatic and internal or external. Manual tools are those that cannot be started automatically, e.g. an EMC test bench for a HW component.

Traceability

Through inherent traceability, WEFACT tracks the status of requirements continuously. Based on the consistent and, if necessary, staged structure of requirements and the execution status of the associated processes, WEFACT is able to determine which processes still need to be run or to be re-run after a modification.

A more detailed description about using the WEFACT user interface is contained in the Handbook for WEFACT [37].

How WEFACT Supports Multiconcern Assurance

WEFACT itself is a workflow tool and not an assurance tool. It provides capabilities to define the detailed assurance process activities (including respective assurance tools to be started) and to run them.

The process model can be defined within the WEFACT user interface or imported from EPF-C reading its UMA output. Figure 11 presents the typical way how WEFACT is intended to be used in the AMASS context.

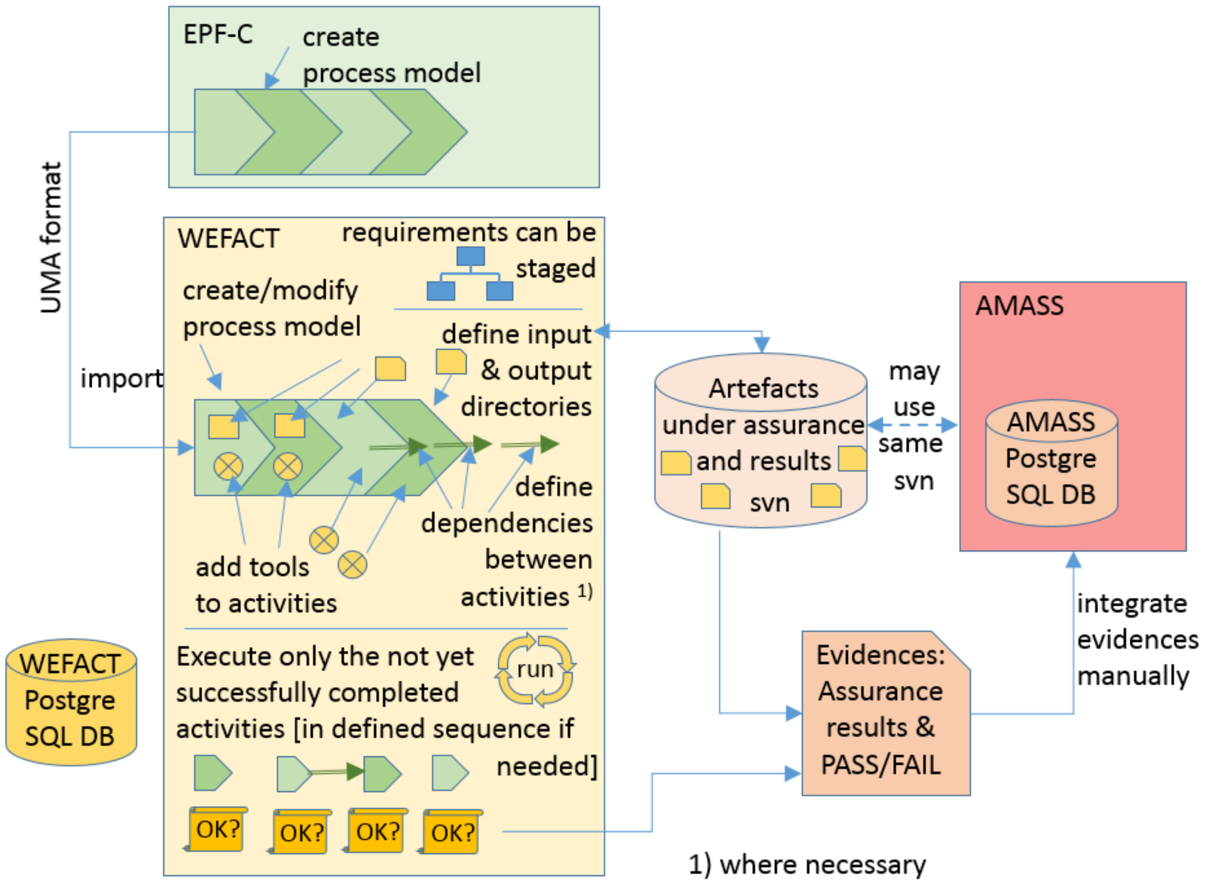

EPF-C
create process model
UMA format
import
WEFACT
requirements can be staged
create/modify process model
define input & output directories
define dependencies between activities 1)
add tools to activities
Execute only the not yet successfully completed activities [in defined sequence if needed]
run
OK?
OK?
OK?
OK?
WEFACT Postgre SQL DB
Artefacts under assurance and results svn
may use same svn
AMASS
AMASS Postgre SQL DB
integrate evidences manually
Evidences: Assurance results & PASS/FAIL
1) where necessary

Figure 11. Typical use of WEFACT in AMASS

As mentioned, the process model can be modified in WEFACT, and the activities defined in the process model are implemented by assigning (and providing) a tool to perform the activity, including the input and output artefacts in the respective directories. If necessary, dependencies between activities can be defined (i.e. their sequence: e.g. an activity can be performed only after another activity has been completed successfully).

WEFACT maintains consistent links between requirements, process activities and all affected artefacts, allowing full traceability. Moreover, WEFACT stores the status of the requirements, which is set to FULFILLED when the associated activities are performed successfully (PASS). On the other hand, changes in system artefacts or requirements are recorded by WEFACT and the status of the respective (associated) requirements is reset. By this mechanism, WEFACT controls, after changes, which activities need to be re-executed in order to restore the assurance status of the system.

After running an activity, the results (output files) are stored in the SVN directory associated with the activity, and the requirement is set according to the result (PASS or FAIL). A "PASS" result represents an evidence for the respective sub-goal in the GSN argumentation of the AMASS assurance case editor. Currently (October 2018), the transfer of the evidence into the argument has to be done manually, i.e. by using the assurance case editor.

In WEFACT, activities can be combined in order to construct multi-concern functionalities. This doesn't require a specific multiconcern WEFACT tool feature but can be implemented by using the standard WEFACT functionalities for assigning tools, which treat (e.g. analyse or test) different quality attributes.

As an example, an activity can be defined calling a security analysis tool; AIT has tried this out with the Microsoft Threat Analysis tool. Similarly, another activity calling a FMEA or a HAZOP tool can be defined in WEFACT to implement the safety analysis part. Also in WEFACT, the (multiconcern) requirement demanding a security-aware HARA can be subdivided into a sub-requirement demanding a security-related analysis and another one for the traditional safety-related hazard analysis. By linking the two aforementioned activities (in WEFACT) to these two sub-requirements, we have realized a combined safety and security analysis. The multiconcern process could be extended by inserting a subsequent interaction point activity, which depends on the successful execution of the two above mentioned (parallel) analyses. This activity can be designed such that contradictions between safety or security-oriented mitigation measures and requirements related to the other quality attribute lead to a FAIL result for the entire composed process, and the (combined) safety AND security analysis requirement is not FUFILLED until a re-iteration of the analyses end up with a non-conflicting set of mitigation measures.

2.3.6. Safety Architect and Cyber Architect

Safety Architect (SA) [14] is initially dedicated to perform classical FTA by generating FT from system model and failure condition analysis of model components. Thanks to the MERgE project [16] the classical FT can be enriched with malicious events which can be caused by an attacker. An example of SA FT extended with a malicious event is shown in Figure 12.

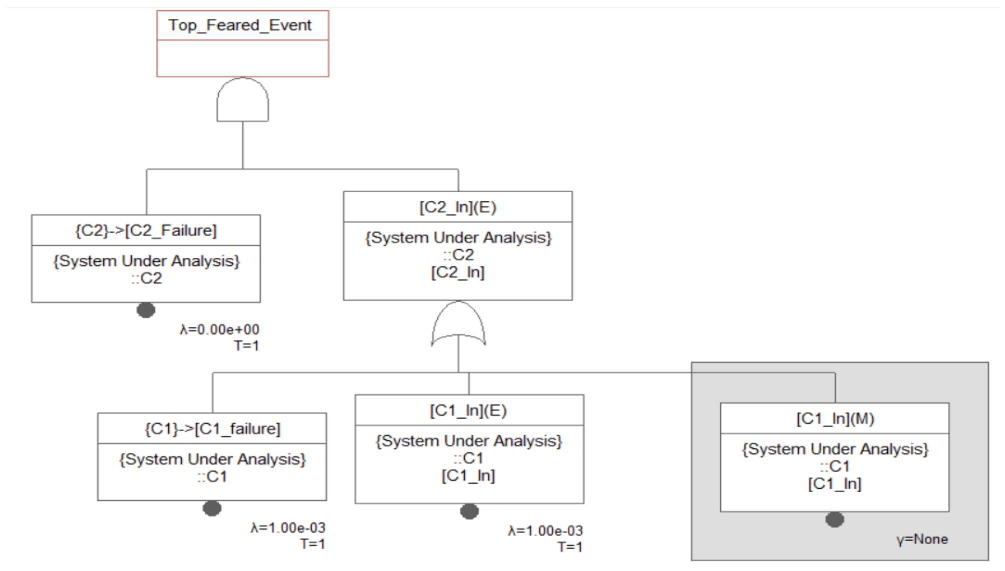


Figure 12. Example of SA FT exported in Arbre Analyste [28]

As illustrated in Figure 12, the SA FT is composed by:

- a top event (e.g., “Top_Feared_Event”),
- intermediate events (e.g., “[C2_In](E)” that represents an erroneous stage of the input of system component 2),
- gates (e.g., AND, OR and others not represented in this example),
- basic events, which can be:
  - safety viewpoint basic events (e.g., “[C1_In](E)” that represents an erroneous state of the input of system component 1)
  - security viewpoint basic event (e.g., “[C1_In](M)” that represents a malicious event caused by an attacker by exploiting the vulnerability of the input of system component 1).

Cyber Architect (CA) [15] is a security analysis tool based on the EBIOS method (Expression des Besoins et Identification des Objectifs de Sécurité - Expression of Needs and Identification of Security Objectives) [17] used to assess and treat risks. The tool implements the five modules of EBIOS method (Module 1 - Study of the context, Module 2 - Study of the feared events, Module 3 - Study of threat scenarios, Module 4 - Study of the risks, Module 5 - Study of the control). Here, we focus on Module 2, whose objective is to systematically identify generic scenarios and feared events that need to be avoided within the study's boundaries. From this systematic analysis, Attack Tree (AT) can be automatically generated in the tool. An example of CA AT is shown in Figure 13.

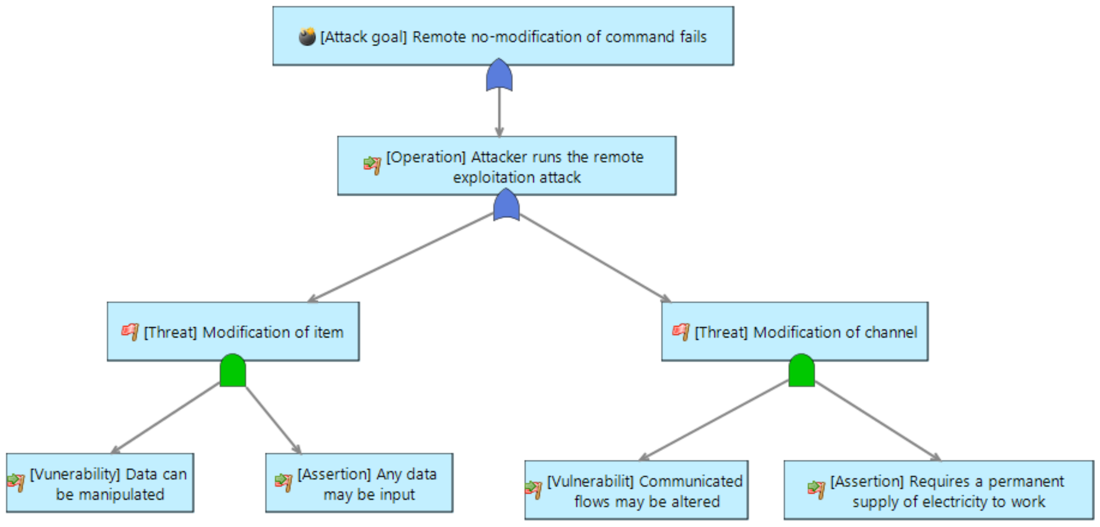

[Attack goal] Remote no-modification of command fails
[Operation] Attacker runs the remote exploitation attack
[Threat] Modification of item
[Threat] Modification of channel
[Vunerability] Data can be manipulated
[Assertion] Any data may be input
[Vulnerabilit] Communicated flows may be altered
[Assertion] Requires a permanent supply of electricity to work

Figure 13. Example of CA AT

The CA AT is composed by:

- a cyber-attack goal (e.g., “Remote no-modification of command fails”),
- an operation event which is an intermediate event or action that can be performed by a system operator (involuntarily) or an attacker (deliberately, e.g. “Attacker runs the remote exploitation attack”),
- gates (e.g., AND, OR),
- threats (e.g., “modification of item”),
- vulnerabilities (e.g., “data can be manipulated”),
- an assertion, which is a statement that represents a condition to be validated in order to consider true a certain branch of the AT (e.g., “any data may be input”).

Difference between SA FT and CA AT. SA FT is a logical structure expressing the relationships and dependencies between a high-level top-event and lower level events, while CA AT is a graph that describes the steps of an attack process. A mapping can be done with some elements of SA FT and CA AT. The main difference between the SA FT and CA AT is the “operation event” that can be really performed by an attacker. While in a SA FT (e.g., Figure 12) we only know the probabilities of occurrence of the basic events, in a CA AT the probability of the “operation events”, which are intermediate events in the tree, can be also taken into account.

2.3.7. Papyrus for Safety and Security Engineering

Papyrus for Safety and Security Engineering (Papyrus SSE) is a framework developed by the CEA to support systems engineering from early phases of the development cycle. Figure 14 shows a two-dimensional workflow. The horizontal axis shows that Papyrus SSE is customized according to the standards from where the fundamental concepts, requirements, and analysis methods are extracted and implemented. The vertical axis shows that Papyrus SSE provides an environment to support several phases of systems design ranging from requirements capturing up to the analysis (validation, verification, tests) of safety and security aspects.

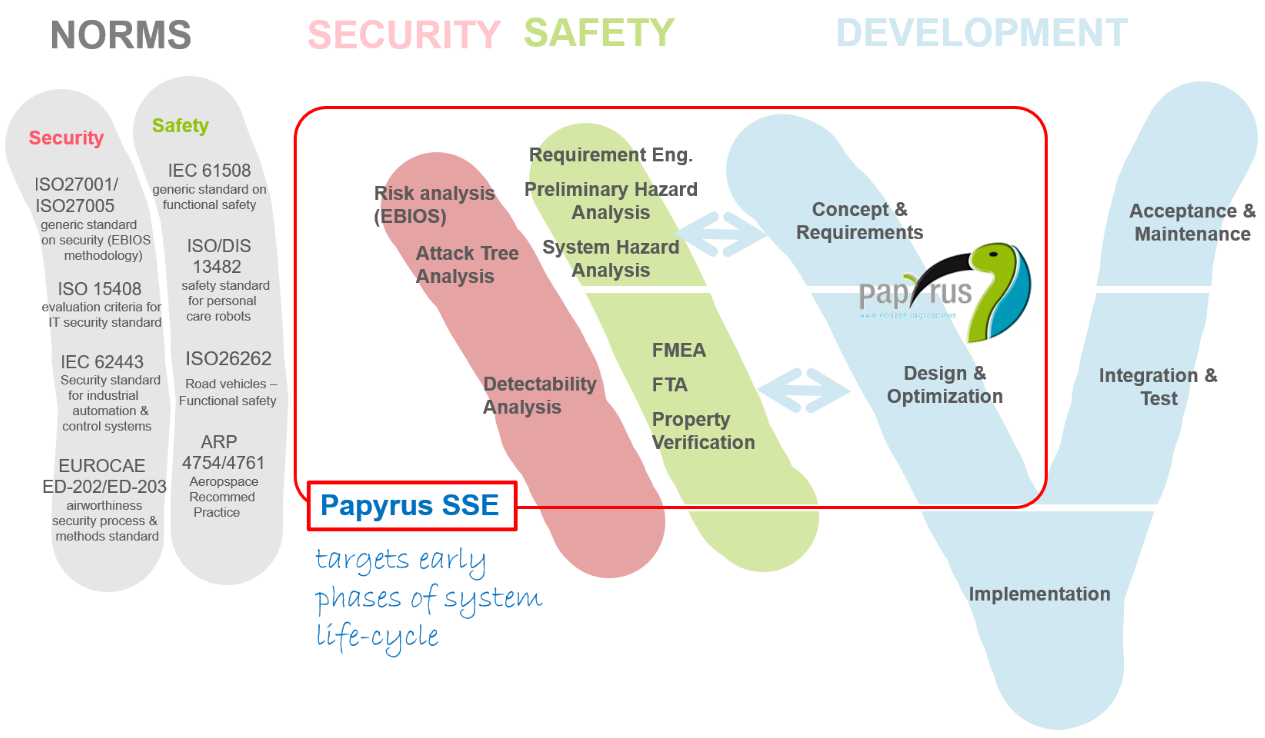

NORMS
SECURITY
SAFETY
DEVELOPMENT
Security
ISO27001/ ISO27005 generic standard on security (EBIOS methodology)
ISO 15408 evaluation criteria for IT security standard
IEC 62443 Security standard for industrial automation & control systems
EUROCAE ED-202/ED-203 airworthiness security process & methods standard
Safety
IEC 61508 generic standard on functional safety
ISO/DIS 13482 safety standard for personal care robots
ISO26262 Road vehicles – Functional safety
ARP 4754/4761 Aeropspace Recommed Practice
Risk analysis (EBIOS)
Attack Tree Analysis
Detectability Analysis
Requirement Eng.
Preliminary Hazard Analysis
System Hazard Analysis
FMEA
FTA
Property Verification
Concept & Requirements
Design & Optimization
Implementation
Integration & Test
Acceptance & Maintenance
Papyrus SSE
targets early phases of system life-cycle

Figure 14. Papyrus SSE supports safety and security analyses during early phases of systems engineering

Regarding safety concerns, Papyrus SSE supports typical safety-oriented analyses like Hazard Analysis and Risk Assessment (HARA), Failure Modes and Effects Analysis (FMEA), Fault Tree Analysis (FTA) as well as the formal verification of properties refined from high level requirements. The referred analyses are supported via dedicated profiles which are deployed according to the application-domain and respective standards:

- ISO/IEC 61508, for functional safety of electrical, electronic, and electronically programmable devices [38]
- ISO 26262, for functional safety of road vehicles [39]
- ISO/DIS 13482, for safety of personal care robots [40]
- ARP 4754 [41] and 4761 [42], for safety of aerospace artefacts and airplanes

Regarding security concerns, Papyrus SSE deploys several profiles to support security-oriented analyses. More specifically, the tool has been customized to support the security risk analysis of information technology, cyber-physical and industrial systems. It supports techniques and methods that cover several phases: modelling and analysis of feared events (including requirements), threats scenarios, vulnerabilities, and countermeasures, Attack Trees, and risks calculation. In addition, a dedicated module allows to formally validate the effectiveness of security countermeasures based upon security test cases. The referred analyses are supported in compliance with the following standards:

- ISO 27001/27005, for the security risk management of information technology systems [43]
- ISO 15408, the evaluation criteria for information technology security [44]
- IEC 62443, targeting the security of industrial automation and control systems [45]
- EUROCAE ED-202 [46] and ED-203 [47], for the airworthiness security process and methods, respectively.

3. Methodological Guide

This chapter constitutes the core of the document. It provides guidance for each functionality of the AMASS platform contributing to the AMASS Multiconcern Assurance approach. This guidance is given as a series of workflows illustrating the main activities to be executed by users to apply the approach. In some cases, workflows are graphically represented as UML activity diagrams, in some other cases they are informally given in a textual format.

3.1. Contract-Based Multiconcern Assurance (*)

The system design comprises the specification of contracts for component, as documented in D3.8 [7] and performed by a combination of development engineer and assurance engineer. This activity is supported by the AMASS platform by means of the CHESS modelling language (CHESSML). In particular component contracts can be modelled as formalisation of requirements by the assurance engineer. Moreover, refinement of contracts can also be modelled along the hierarchical architecture of the system.

Concerning the specification of the contract, it must include the specification of the concern(s) addressed; this has to be done according to the concern(s) addressed by the requirements which are formalised by the contract itself, depending of the typology of the property, the safety or security engineer will be the responsible if doing it. The AMASS platform supports the specification and analysis of contracts where assumptions and guarantees are expressed in Linear-time Temporal Logic (LTL), where LTL can be used to express multi-concern contracts, like safety and security.

Figure 15 shows an example about a CHESS FormalProperty *KeepSafeDistance_Req* formalizing a safety requirement; the FormalProperty is represented with its name, its specification (in OCRA in this case) and with its addressed concern (safety). The FormalProperty represents the guarantee of the *KeepSafeDistance* contract. We can image that according to same safety analysis, the requirement formalized by *KeepSafeDistance_Req* is then refined with requirement related to the performance; the formalization of the latter is represented in Figure 15 by the *BrakeTime_Req* FormalProperty. Then BrakeTime_Req is modelled as the guarantee of the BrakeTime contracts. The refinement of the *KeepSafeDistance* contract into the *BrakeTime* contract can also be modelled in CHESS (not shown in Figure 15).

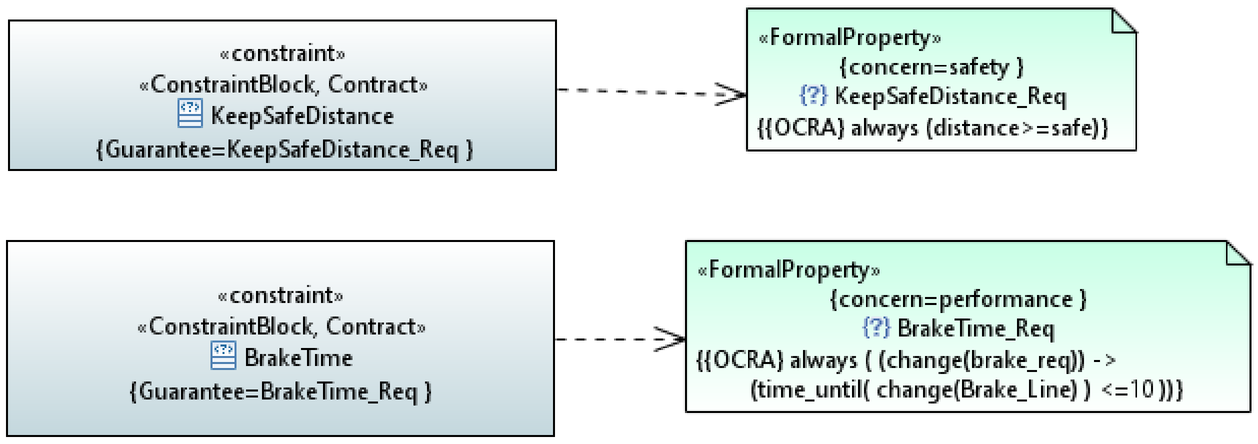

«constraint»
«ConstraintBlock, Contract»
KeepSafeDistance
{Guarantee=KeepSafeDistance_Req }
«FormalProperty»
{concern=safety }
KeepSafeDistance_Req
{(OCRA) always (distance>=safe)}
«constraint»
«ConstraintBlock, Contract»
BrakeTime
{Guarantee=BrakeTime_Req }
«FormalProperty»
{concern=performance }
BrakeTime_Req
{(OCRA) always ((change(brake_req)) ->
(time_until(change(Brake_Line)) <=10))}

Figure 15. Multi-concern contracts

Contracts can then be used for co-analysis, by using the analysis provided by WP3, like the contract refinement analysis, and to enable architecture-driven assurance, so as a way to link the system design to the assurance case.

Figure 16 presents the argument-pattern that is instantiated based on the component contracts, its relations to requirements, the specified refinement as well as evidence supporting confidence in the contracts. The claims “contractKAssume” and “contractKRefine" connect the contract K with other related contracts. The "contractKAssume" connects the contract with other related contracts in its environment. For example, those contracts in the environment that guarantee properties assumed by the contract K. Similarly, the "contractKRefine" connects the contract with the dependent sub-contracts specified by the refinement. In this way, the "contractKAssume" presents the relevant "external" contracts from the environment of a particular component, while "contractKRefine" presents its relevant "internal" contracts, i.e., relevant contracts of its sub-components. Since instantiations of this pattern for different requirements may belong to argument modules for different concerns, the established connections between different contract-specific claims (for which away goals are used) display the interaction points between different concerns. The concern information is embedded in the resulting argument-fragment in the id of its contract-specific claims. When generating a multi-concern argument that a component meets all of its requirements, we can then distinguish between the parts of the argument that belong to different concerns. Similarly, we can generate concern-specific arguments, which would include only those parts associated with a particular concern. That means that the pattern instantiation assuring a safety requirement would include its related both safety and contracts associated with other concerns such as security or performance. Further assurance of those contracts would again not be limited to a single concern but all relevant contracts.

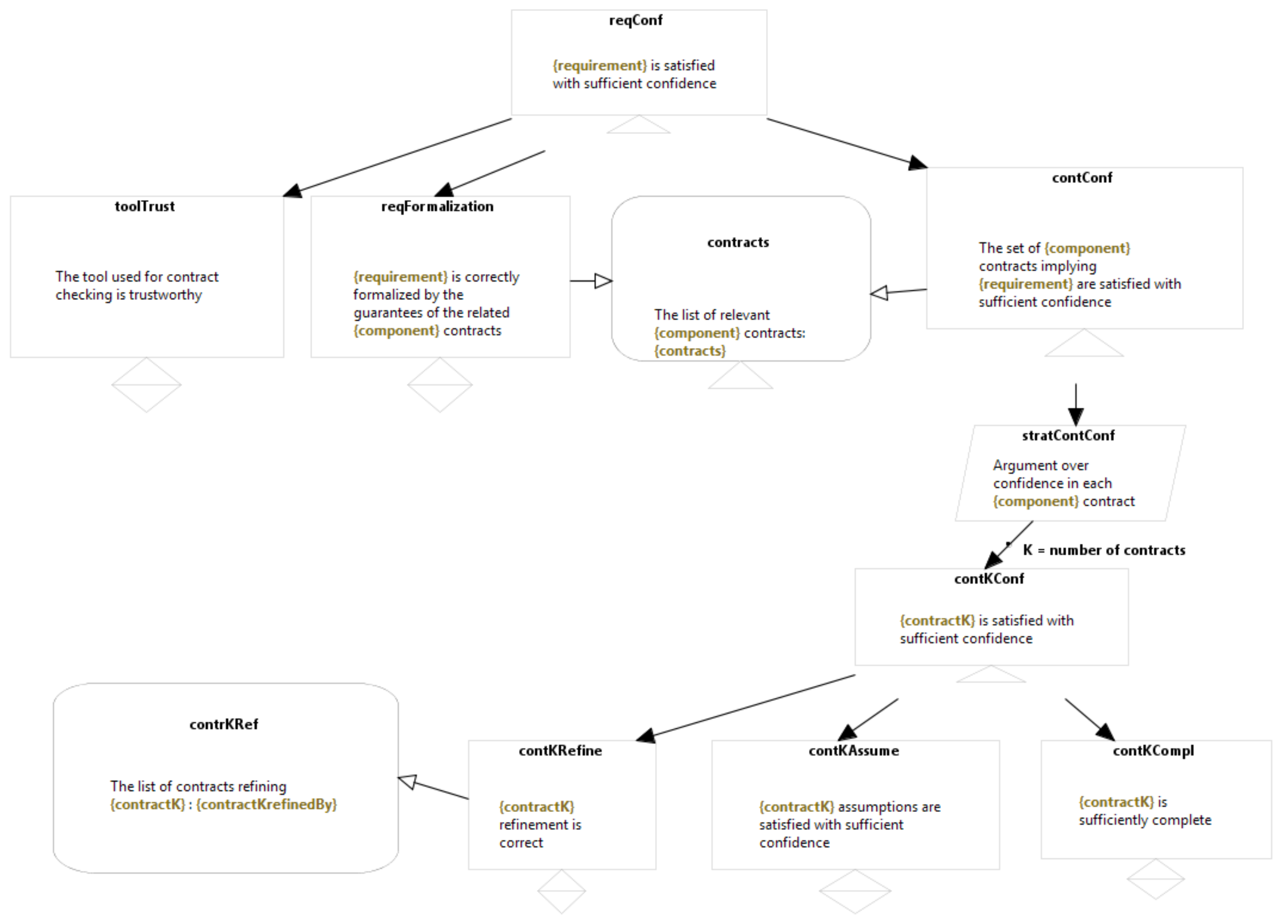

reqConf
{requirement} is satisfied with sufficient confidence
toolTrust
The tool used for contract checking is trustworthy
reqFormalization
{requirement} is correctly formalized by the guarantees of the related {component} contracts
contracts
The list of relevant {component} contracts: {contracts}
contConf
The set of {component} contracts implying {requirement} are satisfied with sufficient confidence
stratContConf
Argument over confidence in each {component} contract
K = number of contracts
contKConf
{contractK} is satisfied with sufficient confidence
contrKRef
The list of contracts refining {contractK} : {contractKrefinedBy}
contKRefine
{contractK} refinement is correct
contKAssume
{contractK} assumptions are satisfied with sufficient confidence
contKCompl
{contractK} is sufficiently complete

Figure 16. The argument pattern for contract-based requirements assurance

For example, by considering the refinement of the contracts, it is possible in the associated argumentation to add information about their dependencies, for instance a contract about safety could be refined by a contract related to performance (e.g. as in Figure 15) or security. Moreover, the argumentation should state about contracts compatibility, e.g. by using the contract-based analysis provided by WP3 and so the produced evidences.

3.1.1. Contract-based Trade-off Analysis in Parameterized Architectures

As shown in Figure 17, the contract-based trade-off analysis requires as input the target parameterized architecture and the set of configurations used to instantiate the parameterized architecture. The methodological guide to parameterize an architecture and to instantiate it is described in the D3.8 [7].

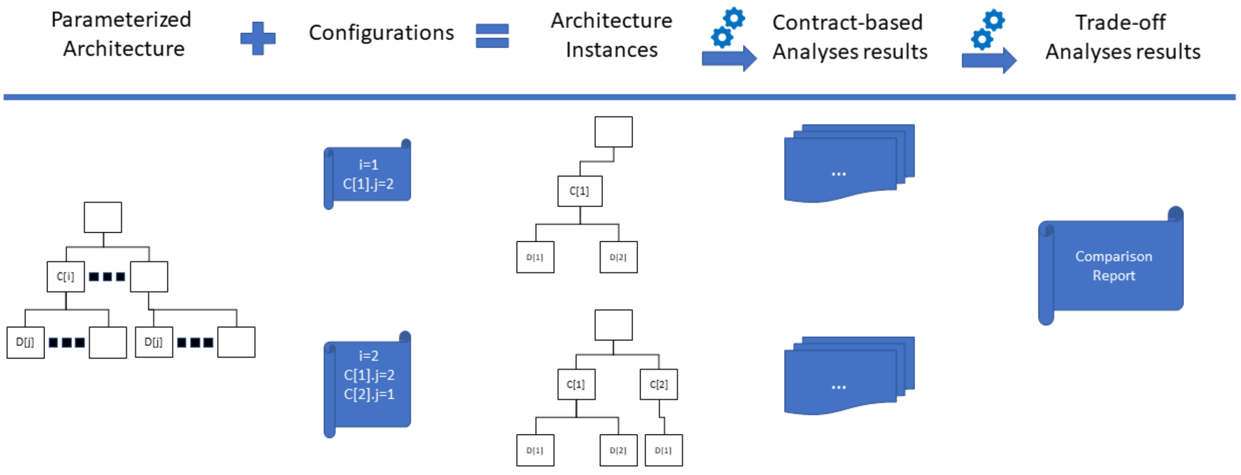


Figure 17. Contract-based Trade-off Analysis takes in input the parameterized architecture and a set of configurations. The process is decomposed in 2 sub-processes; the execution of contact-based checks for each instantiated architecture, and the visualization of the compared results of the checks.

The steps the user must follow are itemized below:

1. The user selects the root component of the parameterized architecture.
2. From the CHESS popup menu, the user executes the contract-based trade-off analysis command.
3. A popup appears showing the options related to the command.
4. The user selects, among the available configurations, the ones he/she wants to analyse.
5. The user selects the contract-based checks, he/she wants to perform (at the release date of this document the check contract refinement is the only check supported).
6. The user confirms the inputs.
7. A dedicated view shows the output of the contract-based trade off analysis in a tabular representation. Columns are the checks, rows are the contracts of each configuration grouped by concern type (safety, security and performance). Each cell shows the result of one check for one configuration.

3.2. Dependability Assurance Case Modelling (*)

The system dependability Assurance Case Modelling workflow is depicted in Figure 18. This workflow describes the work that should be conducted by using the Assurance Case editor provided by OpenCert.

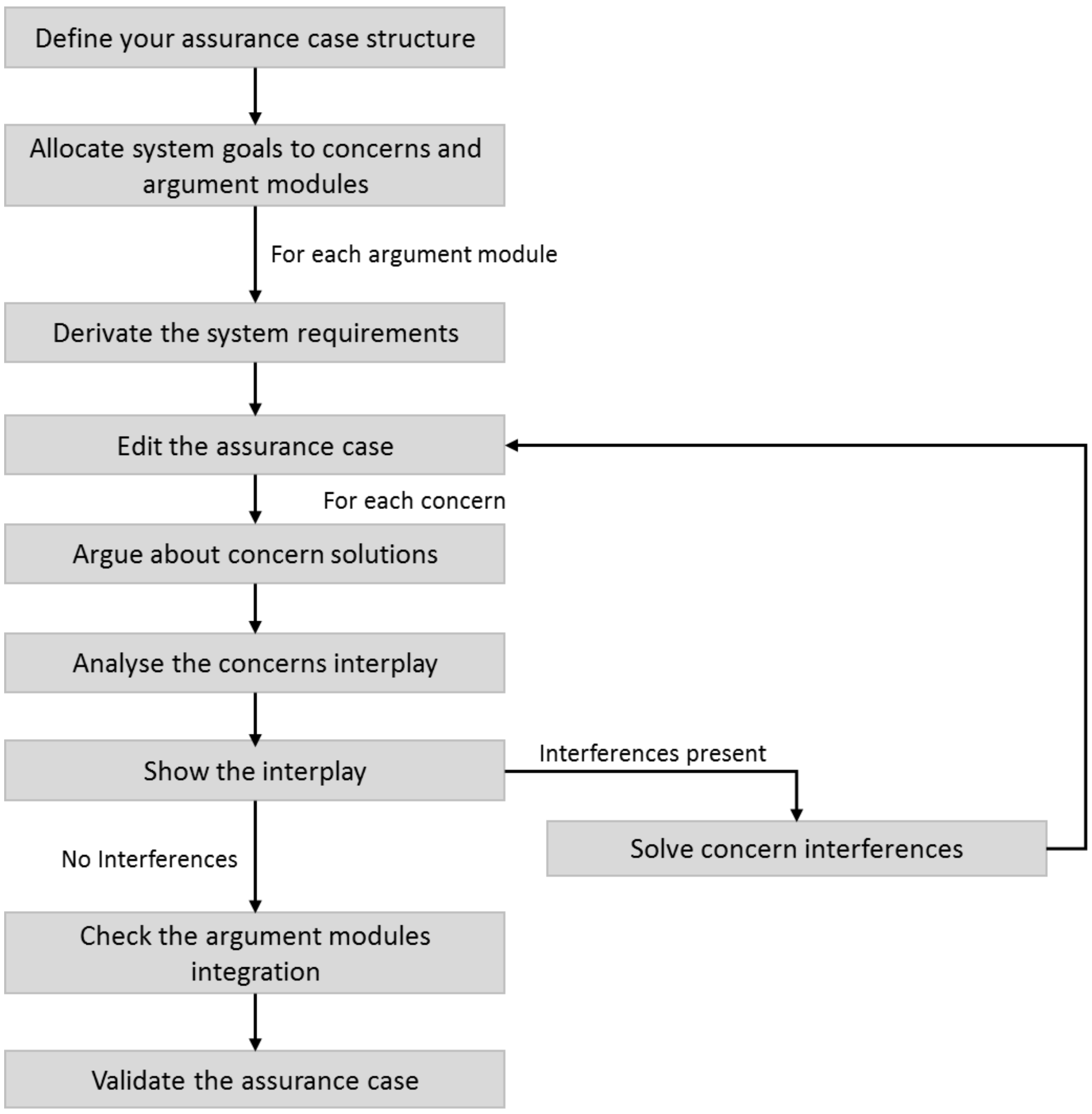

Define your assurance case structure
Allocate system goals to concerns and argument modules
For each argument module
Derivate the system requirements
Edit the assurance case
For each concern
Argue about concern solutions
Analyse the concerns interplay
Show the interplay
Interferences present
Solve concern interferences
No Interferences
Check the argument modules integration
Validate the assurance case

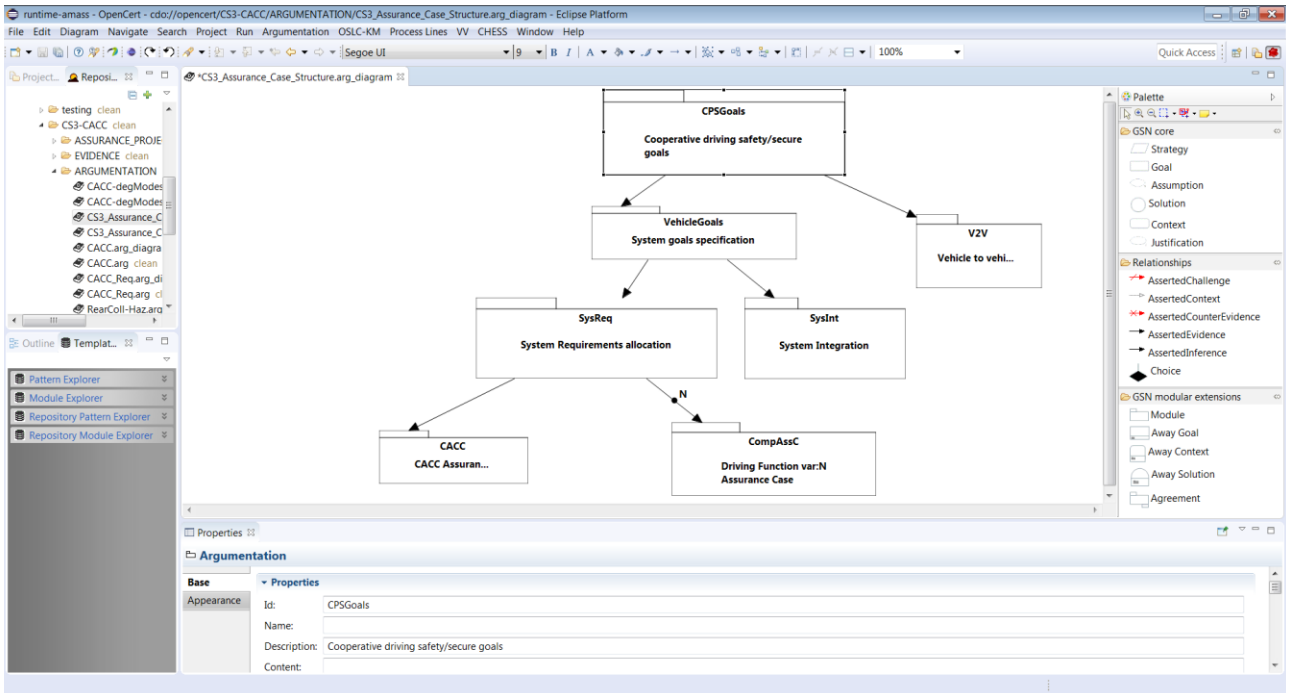
Figure 18. Workflow for Dependability Assurance Case modelling

The system should be assured for dependability properties such that a set of system goals is developed for all the different properties or concerns (safety, security, performance, availability, etc.). The concern-specific goals are the basis for the concern-specific assurance informed of other properties, e.g., security-informed safety assurance. The system goals are supported by the system requirements, which should take into account the needs for all dependability properties (safety, security, performance, availability, etc.). For example, a system goal may be supported by just safety requirements, just by security requirements or both safety and security requirements. The system requirements are allocated into different components and consequently the system assurance case should take into account the assurance case associated to the different components. Each component assurance case might include supporting evidences for one or more dependability properties. For example, a safety-related argument module of one component may be supported by the security-related argument module of that or some other component. The argument modules are used to encapsulate arguments. Interplay of the concerns on all the levels, where cross concern trade off occurs (goals, requirements, components), is handled in the trade-off module.

Define your Assurance Case architecture

In this first step, the Assurance Manager should create the assurance case diagram taking advantage of the assurance case editor provided in OpenCert. To do that, the user could take advantage of the modular argumentation approaches mentioned in D4.3 [3]. In OPENCOSS, it was mentioned that we can encapsulate arguments associated with one component in a module, or in a set of modules. The strategy here is to encapsulate argumentation into modules for organizational purposes. In the GSN Community Standard [5], it also includes a modular extension for GSN which we support by the OpenCert tool, specifically the assurance case editor.

When defining the assurance case structure, the Assurance Manager is also defining the system level goals to be fulfilled.



Properties

Id: CPSGoals

Name:

Description: Cooperative driving safety/secure goals

Content:

Figure 19. Screenshot of the Assurance Case editor defining the assurance case structure

The way the Assurance Manager structure the assurance case might differ from one to another. The assurance manager could decompose it in a similar way that the system high level architecture, so there will be argument modules per component of the system. Or the Assurance Manager could choose to decompose the assurance case into the different concerns the system should take into account in the design such as safety, security, reliability, compliance, etc.

It is highly recommendable to include an argument module for the arguments integration. Its content will be discussed in further steps.

Allocate system goals to concerns

The design of a system should achieve certain goals. Allocate the predefined system level goals to the different argument modules and tag the goals to the different concerns. These goals are usually extracted after some preliminary analysis such as HARA (Hazard Analysis and Risk Assessment) or TARA (Threat Analysis and Risk Assessment) are performed. Each of the system level goals is transformed into a claim in the assurance case. The Assurance engineer should identify which goals are defined in relation to one concern or a subset of concerns. The Safety and Security engineer will agree on which goals are responsible for the safety of the system and which ones with the security.

Derive system requirements

For each of the system goals of each of the concerns, the Assurance engineer should derivate the definition system level requirements in order to fulfil those requirements. This step is highly related with previous Section 3.1. Requirements may be formalised as contract properties and then link them to sub-claims by using the argument pattern depicted in Figure 16.

Edit assurance case

The Assurance engineer should edit the assurance case in order to provide arguments that support the created claims. In Figure 20 the six steps method is explained as shown in. This method is used to define the different claims and evidences to support the assurance case.

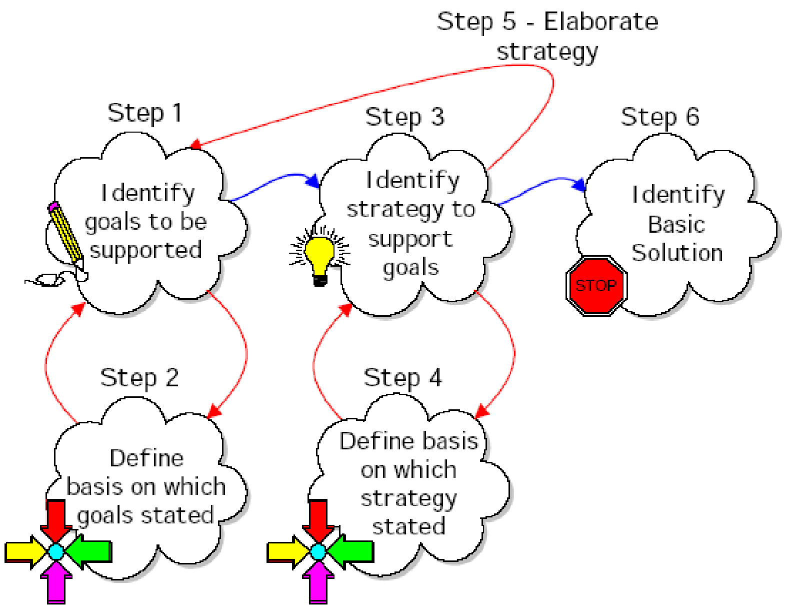
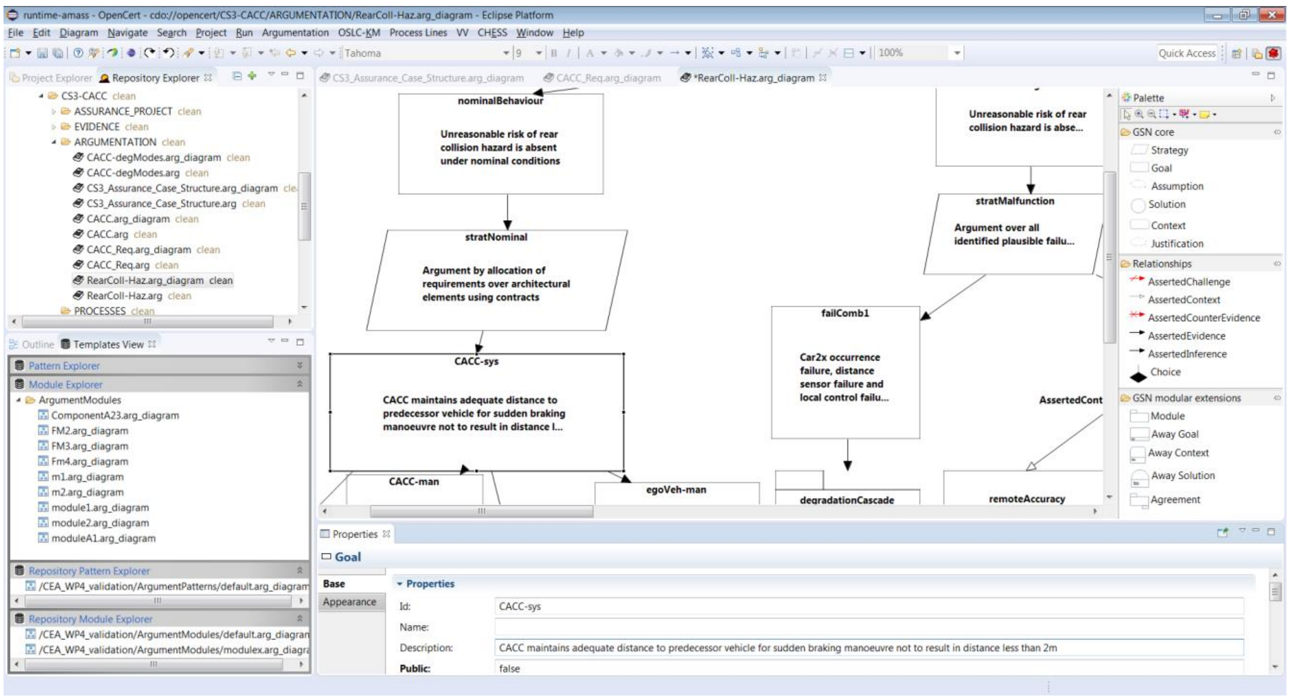

Step 5 - Elaborate strategy
Step 1
Identify goals to be supported
Step 3
Identify strategy to support goals
Step 6
Identify Basic Solution
STOP
Step 2
Define basis on which goals stated
Step 4
Define basis on which strategy stated

Figure 20. Six Step Process for developing goal structures [5]

The assurance case editor provided in OpenCert supports the user in this step. The GSN elements are found on the left hand of the screen in Figure 21. To support the user on the edition, a library of argument patterns can be used. The user can check the available patterns and reusable argument modules on the Templates view.


runtime-amass - OpenCert - cdo://opencert/CS3-CACC/ARGUMENTATION/RearColl-Haz.arg_diagram - Eclipse Platform
Project Explorer
Repository Explorer
CS3-CACC clean
ASSURANCE_PROJECT clean
EVIDENCE clean
ARGUMENTATION clean
CACC-degModes.arg_diagram clean
CACC-degModes.arg clean
CS3_Assurance_Case_Structure.arg_diagram clean
CS3_Assurance_Case_Structure.arg clean
CACC.arg_diagram clean
CACC.arg clean
CACC_Req.arg_diagram clean
CACC_Req.arg clean
RearColl-Haz.arg_diagram clean
RearColl-Haz.arg clean
PROCESSES clean
Outline
Templates View
Pattern Explorer
Module Explorer
ArgumentModules
ComponentA23.arg_diagram
FM2.arg_diagram
FM3.arg_diagram
Fm4.arg_diagram
m1.arg_diagram
m2.arg_diagram
module1.arg_diagram
module2.arg_diagram
moduleA1.arg_diagram
Repository Pattern Explorer
/CEA_WP4_validation/ArgumentPatterns/default.arg_diagram
Repository Module Explorer
/CEA_WP4_validation/ArgumentModules/default.arg_diagram
/CEA_WP4_validation/ArgumentModules/modulex.arg_diagram
CS3_Assurance_Case_Structure.arg_diagram
CACC_Req.arg_diagram
*RearColl-Haz.arg_diagram
nominalBehaviour
Unreasonable risk of rear collision hazard is absent under nominal conditions
stratNominal
Argument by allocation of requirements over architectural elements using contracts
CACC-sys
CACC maintains adequate distance to predecessor vehicle for sudden braking manoeuvre not to result in distance l...
CACC-man
egoVeh-man
Unreasonable risk of rear collision hazard is abse...
stratMalfunction
Argument over all identified plausible failu...
failComb1
Car2x occurrence failure, distance sensor failure and local control failu...
AssertedCont
degradationCascade
remoteAccuracy
Palette
GSN core
Strategy
Goal
Assumption
Solution
Context
Justification
Relationships
AssertedChallenge
AssertedContext
AssertedCounterEvidence
AssertedEvidence
AssertedInference
Choice
GSN modular extensions
Module
Away Goal
Away Context
Away Solution
Agreement
Properties
Goal
Base
Appearance

Properties

Id: CACC-sys
Name:
Description: CACC maintains adequate distance to predecessor vehicle for sudden braking manoeuvre not to result in distance less than 2m
Public: false

Figure 21. Screenshot of the Assurance Case editor editing a claim

Argue about concerns solutions

When editing the assurance case, it is important to argue about the proposed solutions to achieve the previous identified goals. In safety, the Safety engineer should think in the "safety" concept and once the safety concept is designed, it should be translated into safety requirements. Then, the safety requirements should be decomposed into technical requirements in order to progressively and in a traceable manner reach sub-goal by sub-goal the supportive evidence. Similarly, the Security engineer should identify the security assets and security zones. Then after a vulnerability analysis the security engineer should identify the vulnerable assets, which need security protection, and derivate the security solutions. Those proposed solutions should be decomposed into technical security requirements and include them in the assurance case in form of assertions.

Analyse the interplay

When editing the assurance case for each of the concerns, the user (safety or security engineer) typically does not take into account the effect of the decisions in other concerns. In this stage both, the safety and security teams should exchange information and together should analyse the interplay between concerns. Different tools mentioned in Section 3.5, such as FMVEA, can be used to ensure the right analysis is done and the interplay is shown.

Explicitly show the interplay

The interplay or trade-off between the different concerns can be explicitly shown using the "dependency relationship" explained in D4.3 [3]. The Safety and Security engineers should collaborate at this step.

Dependency relationship: The claim A of one attribute depends on the fulfilment of claim B of another attribute. It uses "in the context of" notation with a closed white arrow, as shown in Figure 22 (a).

Conflicting relationship: The assurance measure of attribute A is in conflict with the assurance measure of attribute B. The graphical notation proposed is a red arrow with a slash in the middle as shown in Figure 22 (b). The target of the arrow is a *Claim D* which is conflicting and will become false if the source of the arrow, *Claim C*, becomes true. In a final assurance case, the one produced at delivering the system, this kind of relationships should already be solved. They are useful to exchange the rationale behind the technical decisions.

Supporting relationship: To ensure attribute A, different assurance measure can be used, so if necessary, just one assurance measure can be used to replace other measures if they are considered and addressed individually. The graphical notation used is already present in argument patterns: the choice symbol which is used to represent choices between lines of argumentation used to support a particular claim. For example, *Claim E* is supported either if one of the claims, *Claim F* or *Claim G* is true. It is highly recommended to denote the nature of the choice made for example in Figure 22 (c) where it says "1 out of 2", so one of the claims must be true to support the top claim.

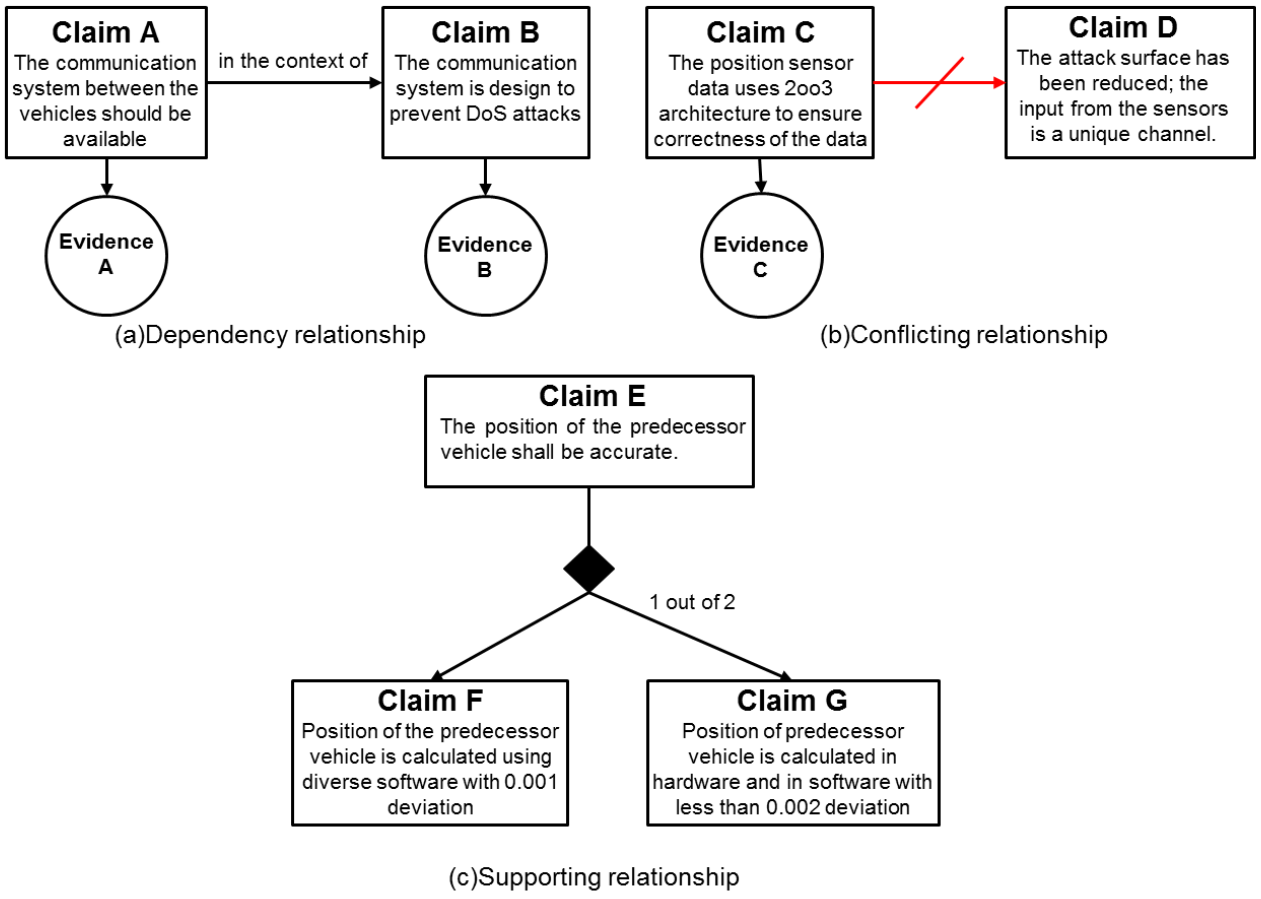

Claim A
The communication system between the vehicles should be available
in the context of
Claim B
The communication system is design to prevent DoS attacks
Evidence A
Evidence B
(a)Dependency relationship
Claim C
The position sensor data uses 2oo3 architecture to ensure correctness of the data
Claim D
The attack surface has been reduced; the input from the sensors is a unique channel.
Evidence C
(b)Conflicting relationship
Claim E
The position of the predecessor vehicle shall be accurate.
1 out of 2
Claim F
Position of the predecessor vehicle is calculated using diverse software with 0.001 deviation
Claim G
Position of predecessor vehicle is calculated in hardware and in software with less than 0.002 deviation
(c)Supporting relationship

Figure 22. Graphical notations used to show the interplay between concerns

A clear view of the impacts reduces the time needed for maintenance and evolution of systems while further guaranteeing safety and security.

Solve the interferences

If a conflicting relationship is identified in the previous step, the arguments involved are not valid. The solutions should be analysed and the assurance case updated once the Safety and Security engineers agree on a solution. Add a justification to the new solution so as when reviewing the assurance case, the rationale behind the deprecated solutions are also listed.

Check argument modules integration

As mentioned in the step "Define your assurance case architecture", when the arguments modules are edited, user should also take into account the integration. The OPENCOSS project did work on this point [74]. Before the system integrator can integrate the architecture components, the system assurance case must first be developed by integrating the argumentation modules belonging to each component comprising the system.

Validate assurance case

In this step the assurance case responsible should ensure that all evidences are traced within the evidence manager. The evidences exist and they provide the expected results to support the goals. At this phase, is advisable that external experts or auditors review the assurance case.

3.3. Process-related Dependability Co-assessment via EPF-C and BVR Tool

In this section, the guide regarding process-related dependability co-assessment is given. More specifically, its workflow is depicted in Figure 23. This workflow describes the work that should be conducted by a process engineer with multi-concern expertise or by a team of process engineers with single-concern expertise for using the integration of EPF Composer and Base Variability Resolution (BVR) Tool in order to perform process-related co-assessment. This workflow is constituted of two inter-related sub-workflows, which can be executed in parallel to some extent.

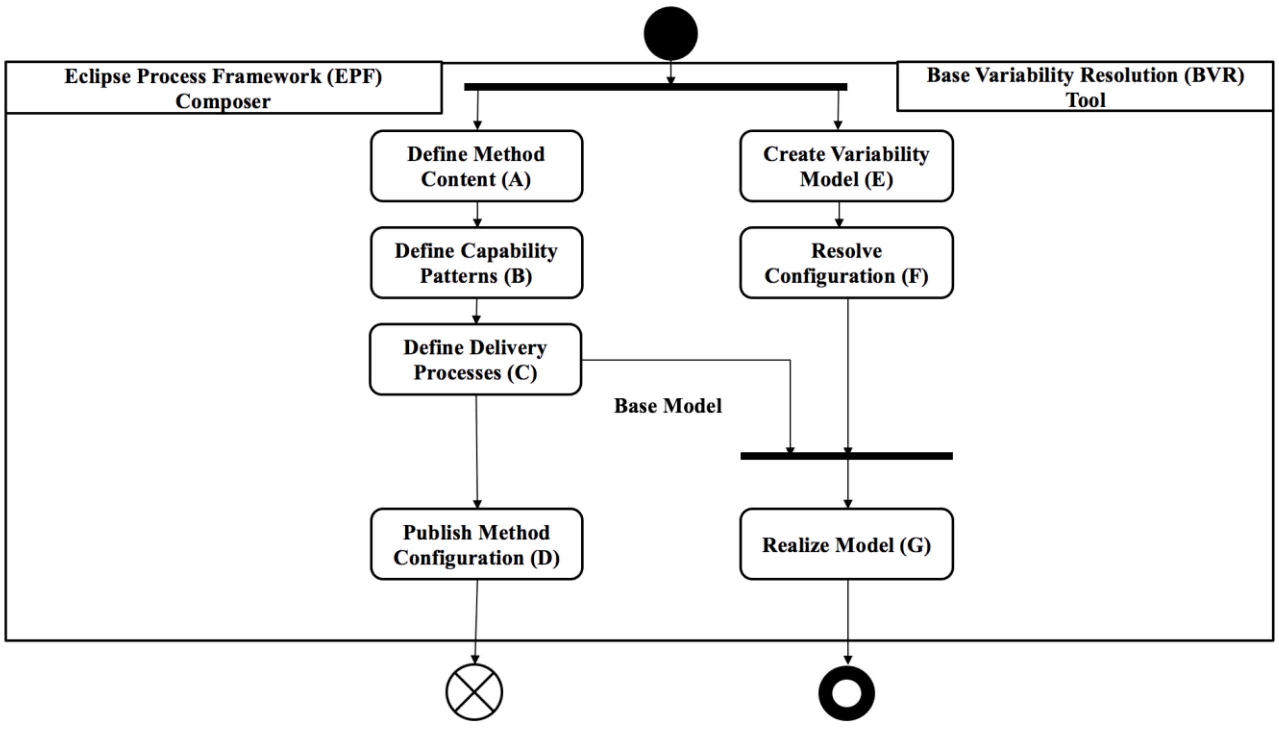

Eclipse Process Framework (EPF) Composer
Base Variability Resolution (BVR) Tool
Define Method Content (A)
Define Capability Patterns (B)
Define Delivery Processes (C)
Publish Method Configuration (D)
Create Variability Model (E)
Resolve Configuration (F)
Base Model
Realize Model (G)

Figure 23. Workflow for System Dependability Co-Assessment

The EPF Composer workflow segment is used to develop and maintain the (software) development process incorporating the dependability requirements through reusable method content which can be methodically combined and organized into process arrangements for a specific project. This allows for exploitation of synergies between safety and security assurance assets by way of reuse. The BVR workflow segment is used to build the multi-concern (software) process line through steps of feature modelling, resolution, mapping of the base model (which has been created through the EPF Composer workflow) and realization of specific models.

A detailed description of the two workflows follows. These are further elucidated using the relevant parts of ISO 26262 [21] and SAE J3061 [22] standards addressing safety and cybersecurity requirements respectively in the case study in Chapter 4.3. The description provided does not include step by step instructions as these are well-covered in the relevant user manuals [19], [20]. The description builds on top of findings, which were presented in [29].

The reason for having chosen normative documents within the automotive domain is twofold: 1) for continuity with D6.3, where cross-concern reuse in the automotive domain was in focus; 2) for internal documents availability and expertise.

3.3.1. EPF Composer Workflow

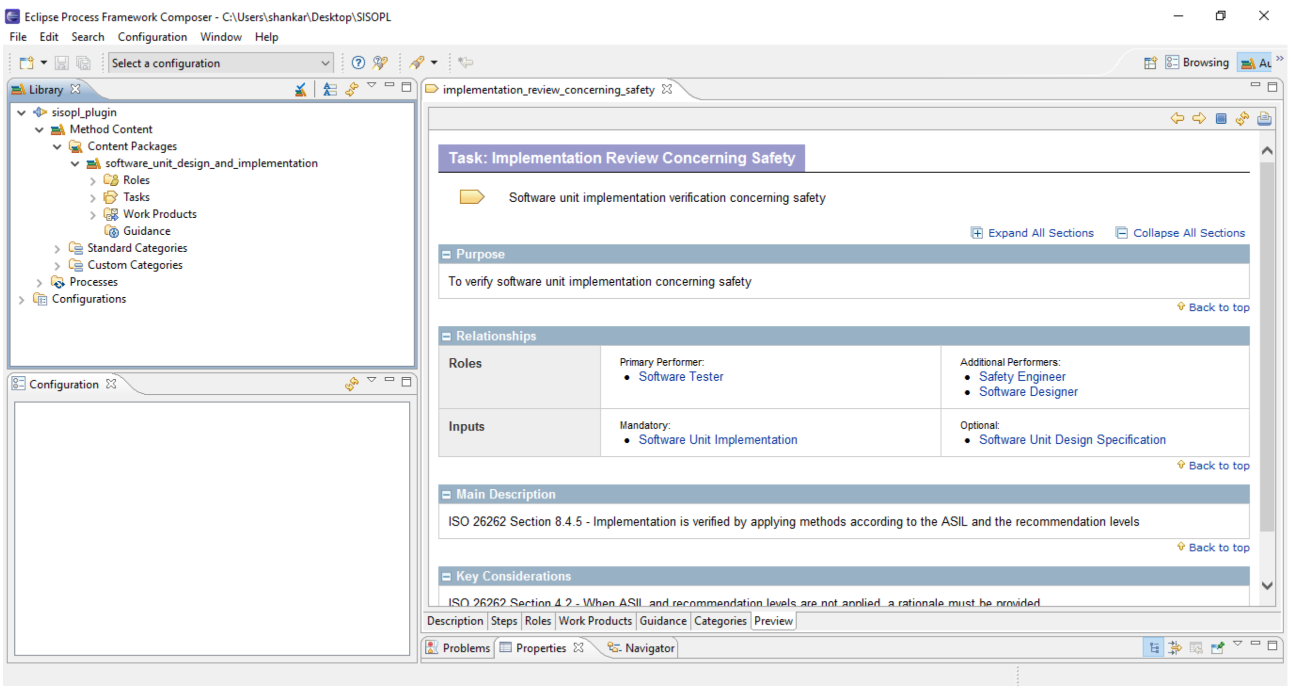
In this subsection, the steps contained in the left-hand side workflow depicted in Figure 23 are explained.

3.3.1.1. Define Method Content (or Process-related Assurance Assets)

This subsubsection explains the step Define Method Content, which corresponds to the Block (A) of the workflow depicted in Figure 23.

As extensively explained in [68] and [69] and also presented in the AMASS User Manual, Method Content consists of Roles, Tasks, Work Products and Guidance. Roles describe who performs the work, Tasks describe how work is performed, Work Products are either what are produced or consumed (Deliverable, Outcome, Artefact) and finally Guidance describes information relevant to the Method Content and is used in the execution of the Tasks. Method Content is defined using the ‘Authoring Perspective’, which is organised in a method plug-in.

Figure 24 depicts the organisation of Method Content and the relationships of its components in the case of ISO 26262 - Software unit implementation verification. More specifically, the task ‘Software unit implementation verification concerning safety’ is performed by the ‘Software Tester’ in a primary role and ‘Safety Engineer’ and ‘Software Designer’ as additional performers. Also, the task has ‘Software Unit Implementation’ as a mandatory input and ‘Software Unit Design Specification’ as optional input.



Task: Implementation Review Concerning Safety

Software unit implementation verification concerning safety

Purpose

To verify software unit implementation concerning safety

Relationships

Roles	Primary Performer: • Software Tester	Additional Performers: • Safety Engineer • Software Designer
Inputs	Mandatory: • Software Unit Implementation	Optional: • Software Unit Design Specification

Main Description

ISO 26262 Section 8.4.5 - Implementation is verified by applying methods according to the ASIL and the recommendation levels

Key Considerations

ISO 26262 Section 4.2 - When ASIL and recommendation levels are not applied, a rationale must be provided

Figure 24. Organization of Method Content

The definition of the Method Content is in itself a composite step that can be depicted as a sequence of sub-steps. Figure 25 depicts such sequence. The details of the various sub-steps follow (see steps from Subsubsubsection 3.3.1.1.1 to Subsubsubsection 3.3.1.1.6).

The steps ‘Create Method Plug-in’ and ‘Create Content Package’ are essentially the first two steps which need to be performed while the remainder steps can also be performed in any other order. The suggested sequence is to minimize switching between various steps thereby offering an efficient work practice.

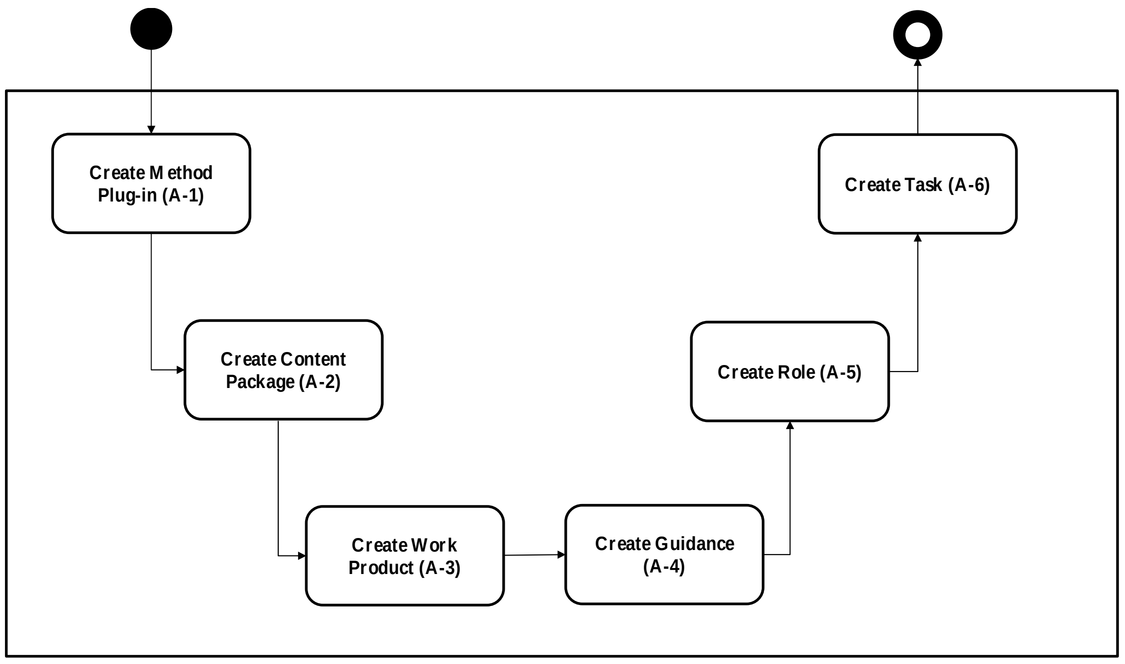
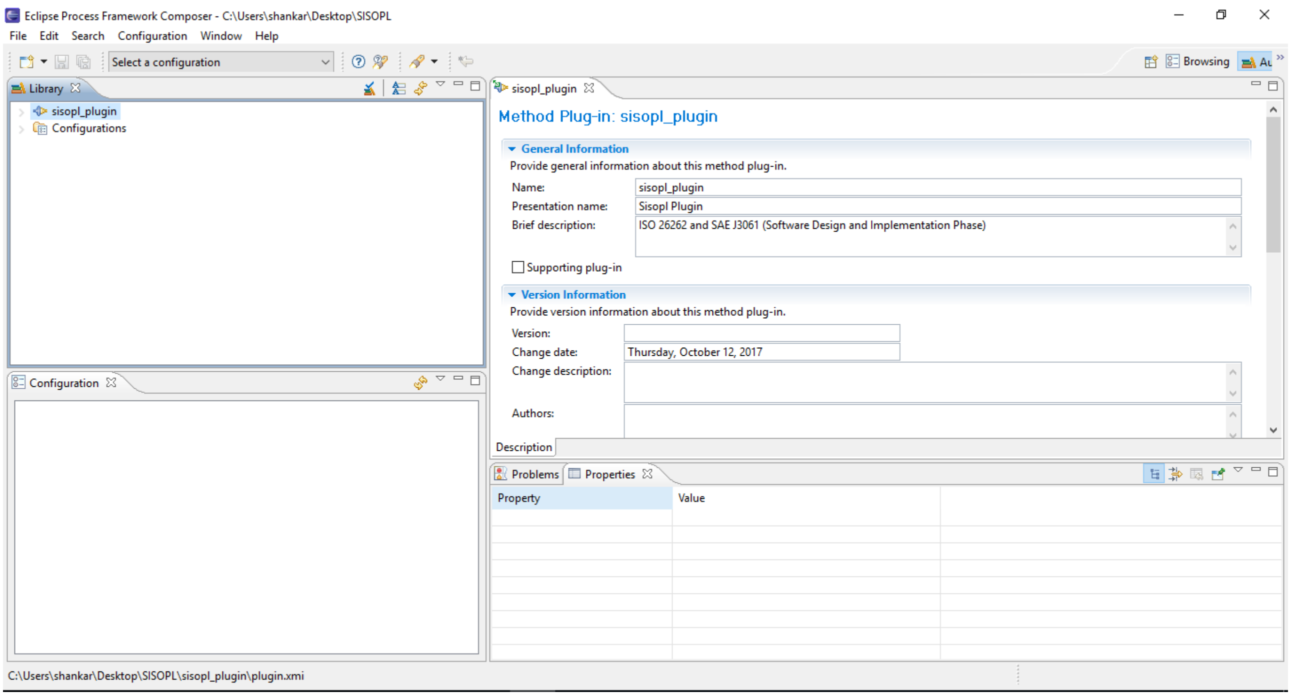

Create Method Plug-in (A-1)
Create Content Package (A-2)
Create Work Product (A-3)
Create Guidance (A-4)
Create Role (A-5)
Create Task (A-6)

Figure 25. Method Content Workflow

3.3.1.1.1. Create Method Plug-in

This corresponds to the Block (A-1) of the workflow depicted in Figure 25. A Method Plug-in contains the Method Content package which is being created. As mentioned earlier, we use the ISO 26262 and SAE J3061 standards in our process model. The created plug-in can be reused in other plug-ins which are created in the future to reuse the content of this plug-in. The Method Plug-in screenshot is depicted in Figure 26. Besides the plug-in description, related change history is maintained providing necessary audit trails.


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL

File Edit Search Configuration Window Help

Select a configuration

Library
- sisopl_plugin
- Configurations

sisopl_plugin

Method Plug-in: sisopl_plugin

General Information

Provide general information about this method plug-in.

Name:	sisopl_plugin
Presentation name:	Sisopl Plugin
Brief description:	ISO 26262 and SAE J3061 (Software Design and Implementation Phase)

☐ Supporting plug-in

Version Information

Provide version information about this method plug-in.

Version:	
Change date:	Thursday, October 12, 2017
Change description:	
Authors:	

Description

Configuration

Problems Properties

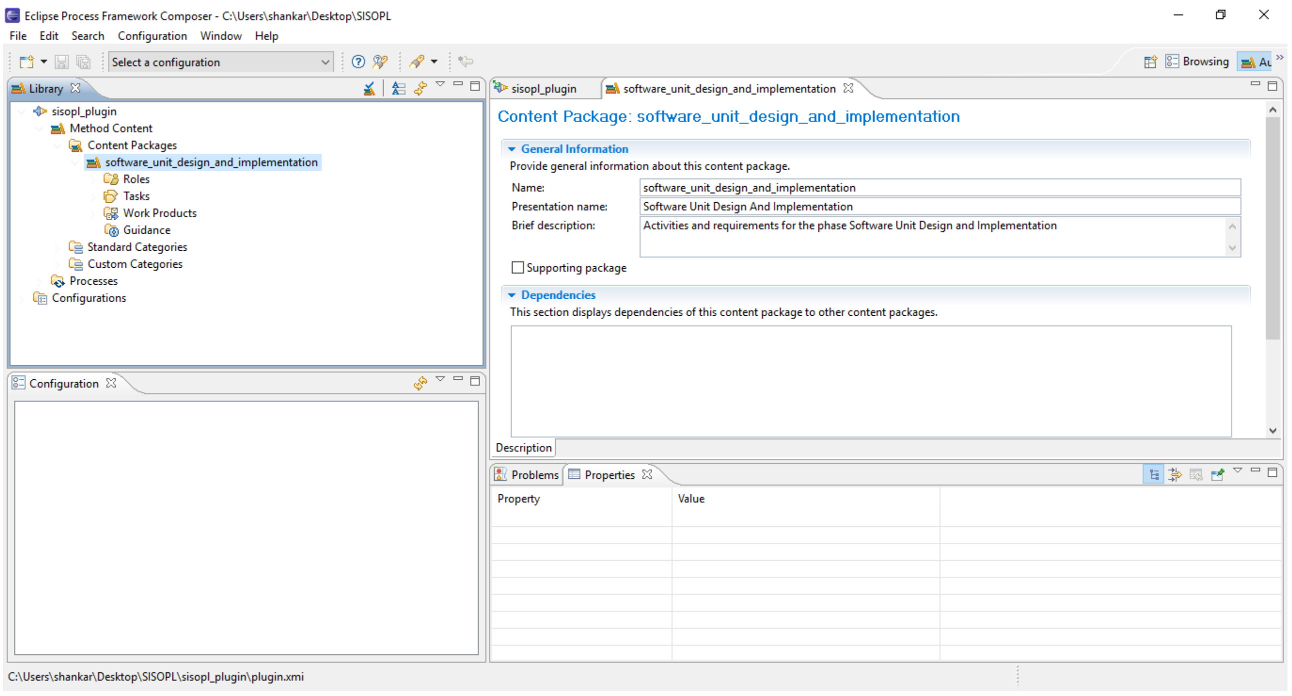
Property	Value

C:\Users\shankar\Desktop\SISOPL\sisopl_plugin\plugin.xmi

Figure 26. Method Plug-in

3.3.1.1.2. *Create Content Package*

This corresponds to the Block (A-2) of the workflow depicted in Figure 25. We next create the Content Package. The content package contains the underlying work products, roles, tasks and guidance. The Content Package screenshot is depicted in Figure 27.


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL

File Edit Search Configuration Window Help

Select a configuration

Library
- sisopl_plugin
  - Method Content
    - Content Packages
      - software_unit_design_and_implementation
        - Roles
        - Tasks
        - Work Products
        - Guidance
    - Standard Categories
    - Custom Categories
  - Processes
- Configurations

sisopl_plugin | software_unit_design_and_implementation

Content Package: software_unit_design_and_implementation

General Information

Provide general information about this content package.

Name:	software_unit_design_and_implementation
Presentation name:	Software Unit Design And Implementation
Brief description:	Activities and requirements for the phase Software Unit Design and Implementation

☐ Supporting package

Dependencies

This section displays dependencies of this content package to other content packages.

Description

Configuration

Problems Properties

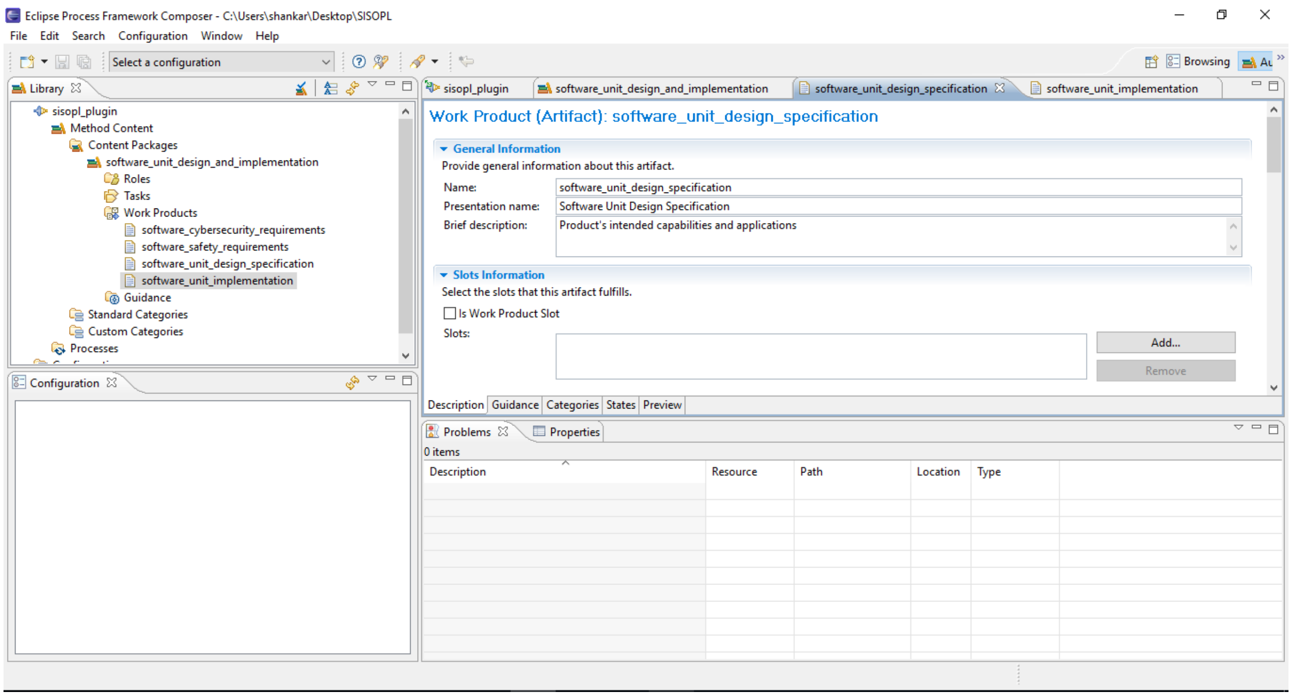
Property	Value

C:\Users\shankar\Desktop\SISOPL\sisopl_plugin\plugin.xmi

Figure 27. Content Package

3.3.1.1.3. *Create Work Product*

This corresponds to the Block (A-3) of the workflow depicted in Figure 25. Three types of Work Products may be created, namely Artefact, Outcome and Deliverable. An Artefact is a tangible Work Product while an Outcome is an intangible Work Product such as a result or a state. A Deliverable is a collection of Work Products which define typical content to be delivered. Screenshots of two Work Products are depicted in Figure 28 (Software Unit Design Specification) and Figure 29 (Software Unit Implementation). In this example, Software Unit Design Specification is an input work product to several tasks while Software Unit Implementation is an output work product of the design tasks.


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL

File Edit Search Configuration Window Help

Select a configuration

Library
- sisopl_plugin
  - Method Content
    - Content Packages
      - software_unit_design_and_implementation
        - Roles
        - Tasks
        - Work Products
          - software_cybersecurity_requirements
          - software_safety_requirements
          - software_unit_design_specification
          - software_unit_implementation
        - Guidance
    - Standard Categories
    - Custom Categories
  - Processes

Configuration

sisopl_plugin | software_unit_design_and_implementation | software_unit_design_specification | software_unit_implementation

Work Product (Artifact): software_unit_design_specification

General Information

Provide general information about this artifact.

Name: software_unit_design_specification

Presentation name: Software Unit Design Specification

Brief description: Product's intended capabilities and applications

Slots Information

Select the slots that this artifact fulfills.

☐ Is Work Product Slot

Slots:

Add... Remove

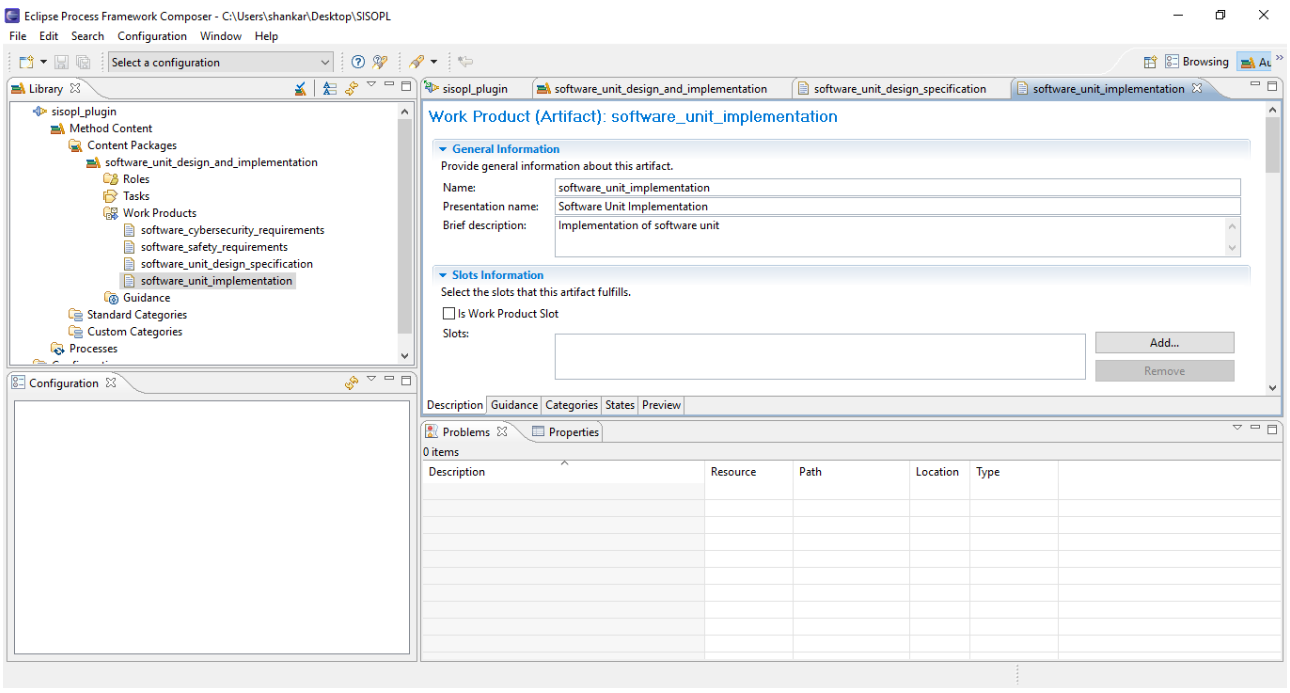
Description | Guidance | Categories | States | Preview

Problems | Properties

0 items

Description	Resource	Path	Location	Type

Figure 28. Work Product 1


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL

File Edit Search Configuration Window Help

Select a configuration

Library
- sisopl_plugin
  - Method Content
    - Content Packages
      - software_unit_design_and_implementation
        - Roles
        - Tasks
        - Work Products
          - software_cybersecurity_requirements
          - software_safety_requirements
          - software_unit_design_specification
          - software_unit_implementation
        - Guidance
    - Standard Categories
    - Custom Categories
  - Processes

Configuration

sisopl_plugin | software_unit_design_and_implementation | software_unit_design_specification | software_unit_implementation

Work Product (Artifact): software_unit_implementation

General Information

Provide general information about this artifact.

Name: software_unit_implementation

Presentation name: Software Unit Implementation

Brief description: Implementation of software unit

Slots Information

Select the slots that this artifact fulfills.

☐ Is Work Product Slot

Slots:

Add... Remove

Description | Guidance | Categories | States | Preview

Problems | Properties

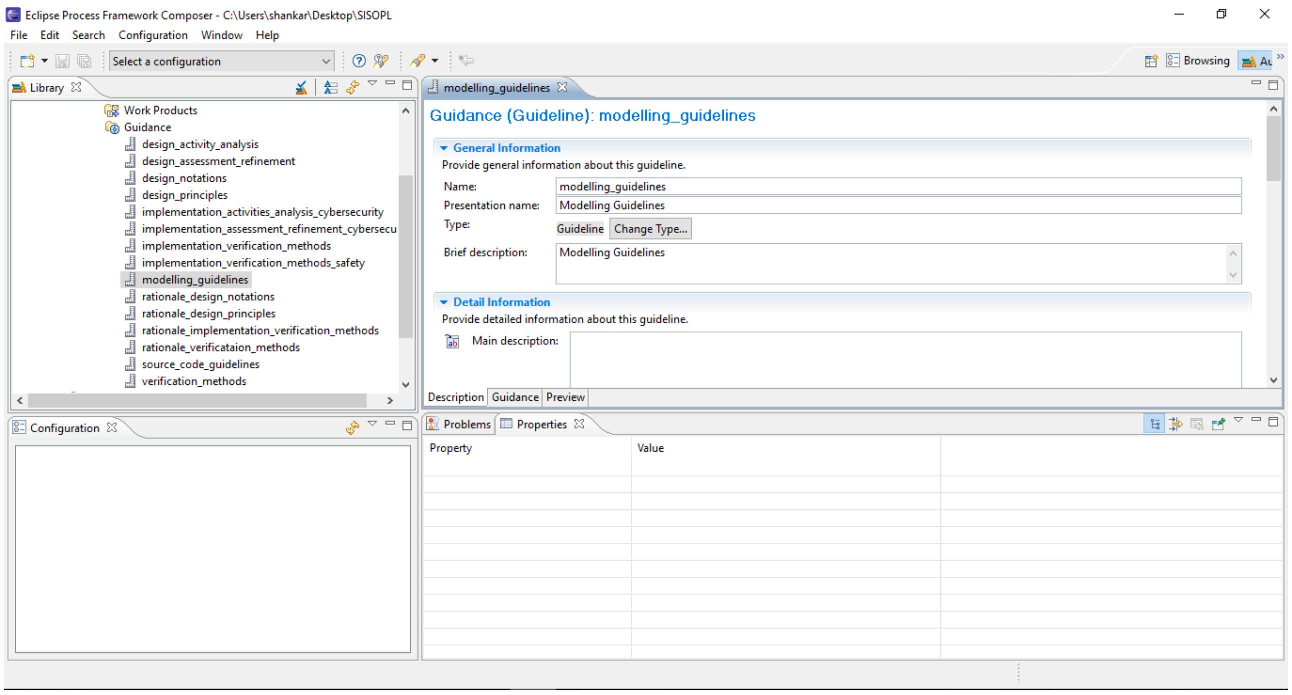
0 items

Description	Resource	Path	Location	Type

Figure 29. Work Product 2

3.3.1.1.4. *Create Guidance*

This corresponds to the Block (A-4) of the workflow depicted in Figure 25. Guidance provides supplementary information for performing the task. Several guidelines are defined and Figure 30 depicts a screenshot of 'Modelling Guidelines'.


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL

Library:
- Work Products
- Guidance
  - design_activity_analysis
  - design_assessment_refinement
  - design_notations
  - design_principles
  - implementation_activities_analysis_cybersecurity
  - implementation_assessment_refinement_cybersecu
  - implementation_verification_methods
  - implementation_verification_methods_safety
  - modelling_guidelines
  - rationale_design_notations
  - rationale_design_principles
  - rationale_implementation_verification_methods
  - rationale_verificataion_methods
  - source_code_guidelines
  - verification_methods

Guidance (Guideline): modelling_guidelines

General Information
Provide general information about this guideline.
Name: modelling_guidelines
Presentation name: Modelling Guidelines
Type: Guideline
Brief description: Modelling Guidelines

Detail Information
Provide detailed information about this guideline.
Main description:

Description | Guidance | Preview

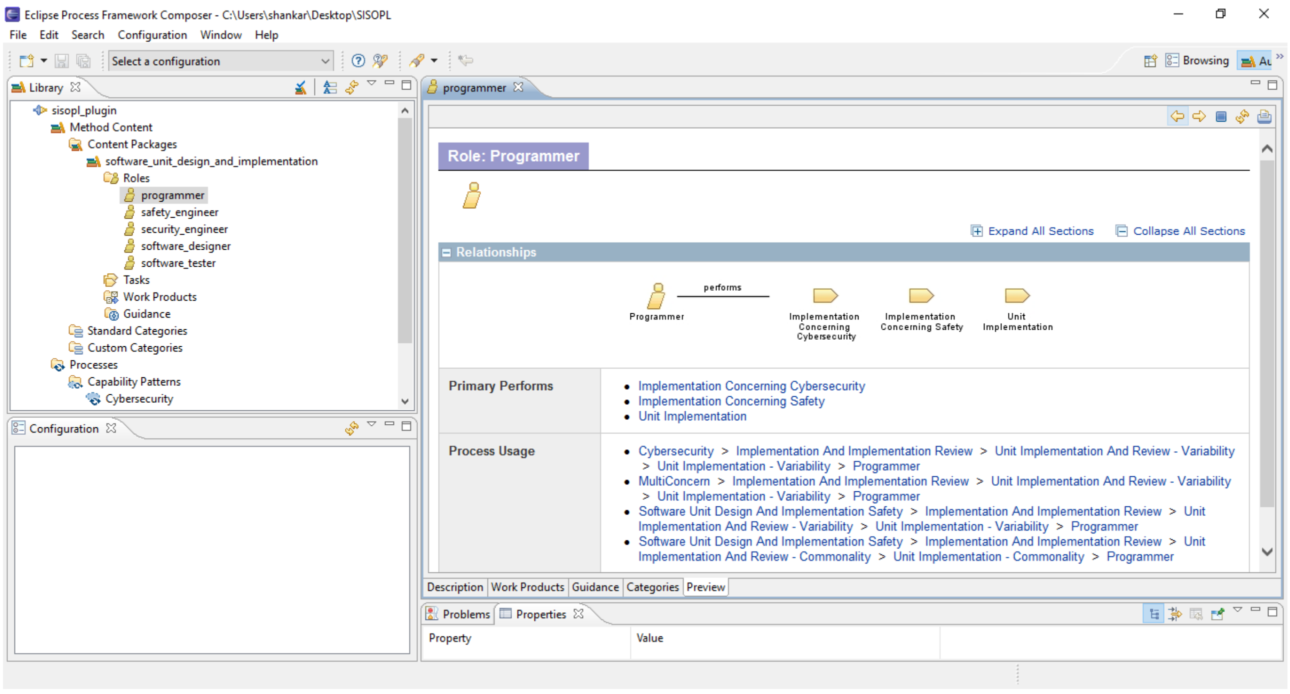
Problems | Properties

Property	Value

Figure 30. Guideline

3.3.1.1.5. *Create Role*

This corresponds to the Block (A-5) of the workflow depicted in Figure 25. Roles define responsibilities for Work Products which are produced, the work to be done by the role and the results to be produced. Figure 31 depicts a screenshot showing the relationship between the created role and tasks.


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL

Library:
- sisopl_plugin
  - Method Content
    - Content Packages
      - software_unit_design_and_implementation
        - Roles
          - programmer
          - safety_engineer
          - security_engineer
          - software_designer
          - software_tester
        - Tasks
        - Work Products
        - Guidance
    - Standard Categories
    - Custom Categories
  - Processes
    - Capability Patterns
      - Cybersecurity

Role: Programmer

Relationships

performs
Programmer
Implementation Concerning Cybersecurity
Implementation Concerning Safety
Unit Implementation

Primary Performs	• Implementation Concerning Cybersecurity • Implementation Concerning Safety • Unit Implementation
Process Usage	• Cybersecurity > Implementation And Implementation Review > Unit Implementation And Review - Variability > Unit Implementation - Variability > Programmer • MultiConcern > Implementation And Implementation Review > Unit Implementation And Review - Variability > Unit Implementation - Variability > Programmer • Software Unit Design And Implementation Safety > Implementation And Implementation Review > Unit Implementation And Review - Variability > Unit Implementation - Variability > Programmer • Software Unit Design And Implementation Safety > Implementation And Implementation Review > Unit Implementation And Review - Commonality > Unit Implementation - Commonality > Programmer

Description | Work Products | Guidance | Categories | Preview

Figure 31. Role and Work Product Relationship

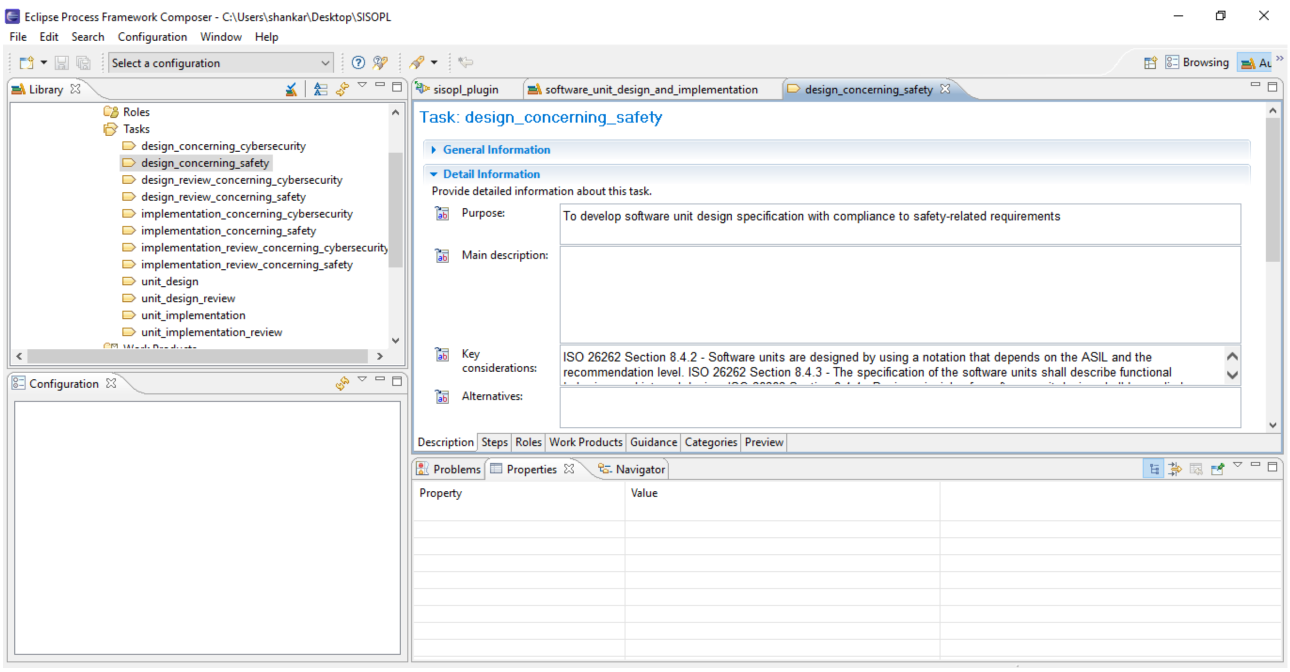
3.3.1.1.6. *Create Task*

This corresponds to the Block (A-6) of the workflow depicted in Figure 25. A Task defines the steps needed to perform the purpose of the task and are related to the Roles defined (who performs the task), Work Products which are either produced or consumed, and any Guidance which may be defined. Tasks may be assigned to one or more roles. For example, in the co-analysis and design of safety and cybersecurity requirements, a task may be assigned to both a 'Safety Engineer' and a 'Security Engineer' who work together. Several Work Products may be specified. Input Work Products may be either mandatory or optional in nature. Tasks are composed of steps which may or may not be performed in a certain order and a step may also be omitted for a particular instance of the task. Figure 32 depicts the screenshot of creation of a Task (Design Concerning Safety), Figure 33 depicts the various steps which makeup the Task and Figure 34 depicts the relationships between the Task with the Roles, Work Products and Guidance.

The task consists of the following three steps:

- Design software units by using notation that depends on ASIL and the recommendation level.
- Describe functional behaviour and internal design in the specification of the software units.
- Apply design principles for software unit design depending on the ASIL and the recommendation levels.

The Task/Role relationship for 'Design Concerning Safety' shows that the task is performed primarily by the Safety Engineer and additionally by the Software Designer. The Task/Work Product relationship shows that the task has 'Software Unit Design Specification' and 'Software Safety Requirements' as mandatory inputs and 'Software Unit Implementation' as output. The Task/Guidance relationship shows that the task uses Design Notations, Rationale Design Notations, Design Principles and Rationale Design Principles as guidance.


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL
File Edit Search Configuration Window Help
Select a configuration
Browsing
Library
Roles
Tasks
design_concerning_cybersecurity
design_concerning_safety
design_review_concerning_cybersecurity
design_review_concerning_safety
implementation_concerning_cybersecurity
implementation_concerning_safety
implementation_review_concerning_cybersecurity
implementation_review_concerning_safety
unit_design
unit_design_review
unit_implementation
unit_implementation_review
Configuration
sisopl_plugin
software_unit_design_and_implementation
design_concerning_safety
Description Steps Roles Work Products Guidance Categories Preview
Problems Properties Navigator
Property
Value

Task: design_concerning_safety

General Information

Detail Information

Provide detailed information about this task.

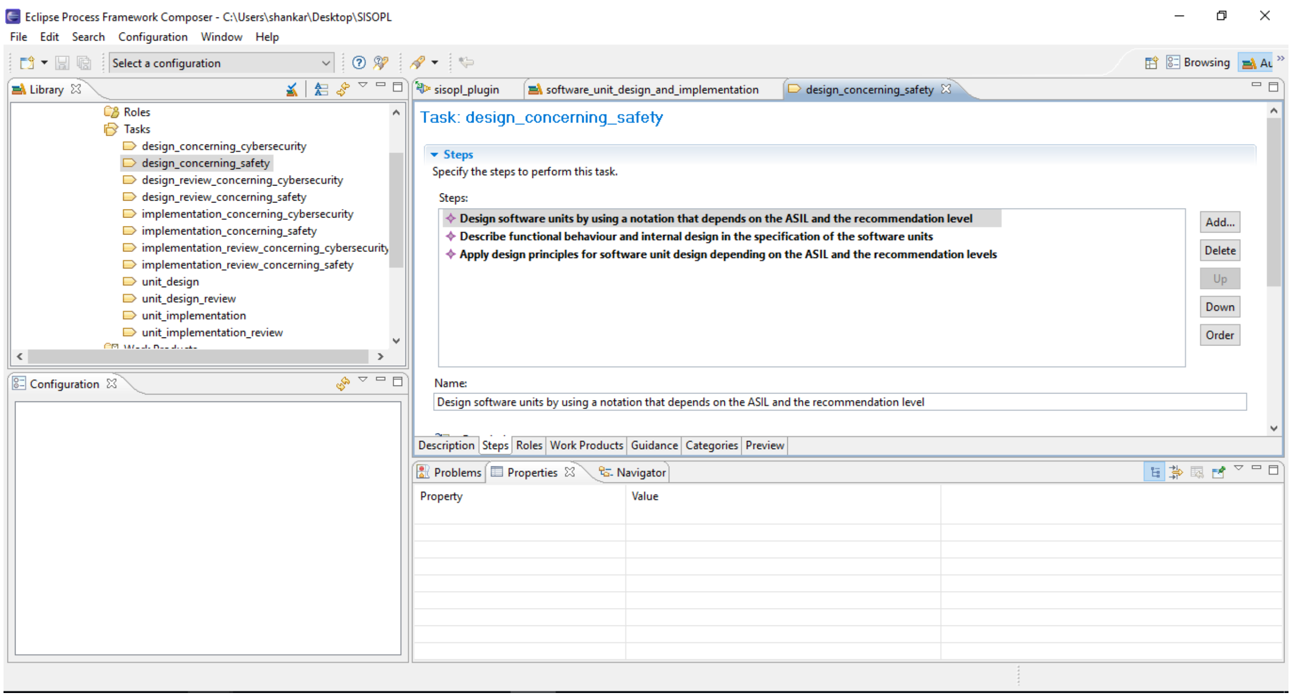
Purpose: To develop software unit design specification with compliance to safety-related requirements

Main description:

Key considerations: ISO 26262 Section 8.4.2 - Software units are designed by using a notation that depends on the ASIL and the recommendation level. ISO 26262 Section 8.4.3 - The specification of the software units shall describe functional

Alternatives:

Figure 32. Create a Task


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL
File Edit Search Configuration Window Help
Select a configuration
Browsing
Library
Roles
Tasks
design_concerning_cybersecurity
design_concerning_safety
design_review_concerning_cybersecurity
design_review_concerning_safety
implementation_concerning_cybersecurity
implementation_concerning_safety
implementation_review_concerning_cybersecurity
implementation_review_concerning_safety
unit_design
unit_design_review
unit_implementation
unit_implementation_review
Configuration
sisopl_plugin
software_unit_design_and_implementation
design_concerning_safety
Add...
Delete
Up
Down
Order
Description Steps Roles Work Products Guidance Categories Preview
Problems Properties Navigator
Property
Value

Task: design_concerning_safety

Steps

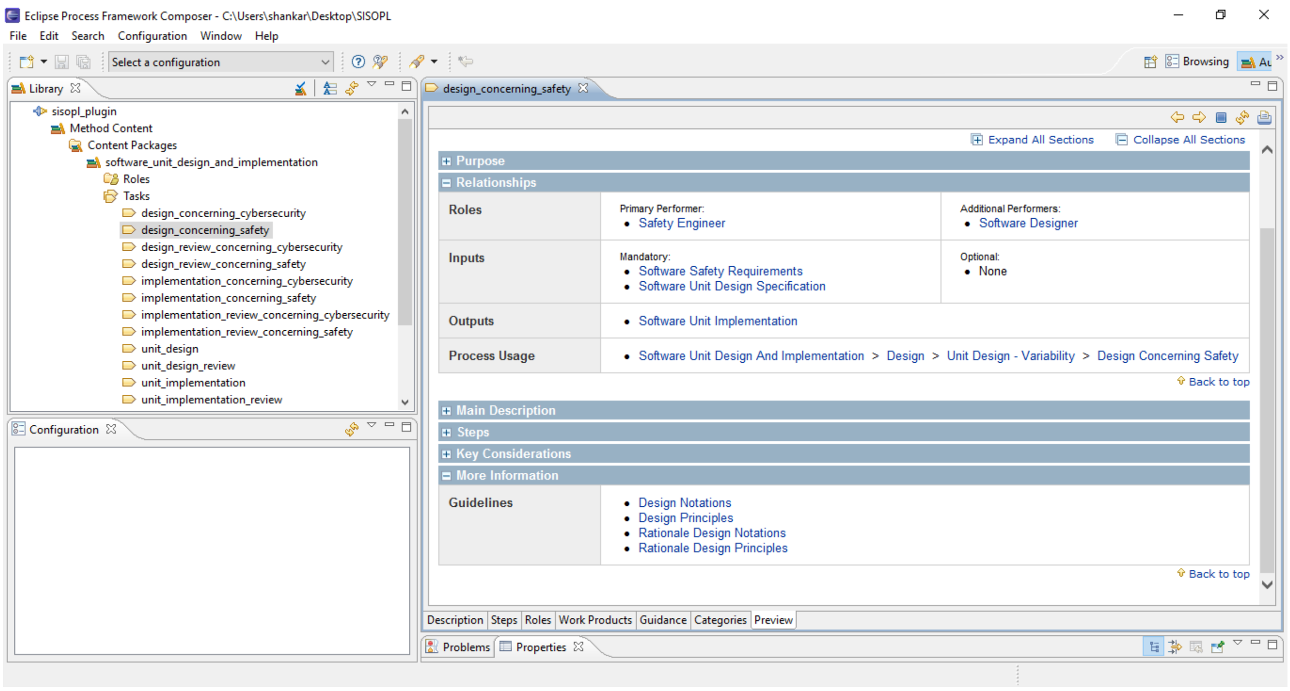
Specify the steps to perform this task.

Steps:

- Design software units by using a notation that depends on the ASIL and the recommendation level
- Describe functional behaviour and internal design in the specification of the software units
- Apply design principles for software unit design depending on the ASIL and the recommendation levels

Name:
Design software units by using a notation that depends on the ASIL and the recommendation level

Figure 33. Task Steps


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL
File Edit Search Configuration Window Help
Select a configuration
Browsing
Library
sisopl_plugin
Method Content
Content Packages
software_unit_design_and_implementation
Roles
Tasks
design_concerning_cybersecurity
design_concerning_safety
design_review_concerning_cybersecurity
design_review_concerning_safety
implementation_concerning_cybersecurity
implementation_concerning_safety
implementation_review_concerning_cybersecurity
implementation_review_concerning_safety
unit_design
unit_design_review
unit_implementation
unit_implementation_review
Configuration
design_concerning_safety
Expand All Sections
Collapse All Sections
Back to top
Description Steps Roles Work Products Guidance Categories Preview
Problems Properties

Purpose

Relationships

Roles	Primary Performer: • Safety Engineer	Additional Performers: • Software Designer
Inputs	Mandatory: • Software Safety Requirements • Software Unit Design Specification	Optional: • None
Outputs	• Software Unit Implementation	
Process Usage	• Software Unit Design And Implementation > Design > Unit Design - Variability > Design Concerning Safety	

Main Description

Steps

Key Considerations

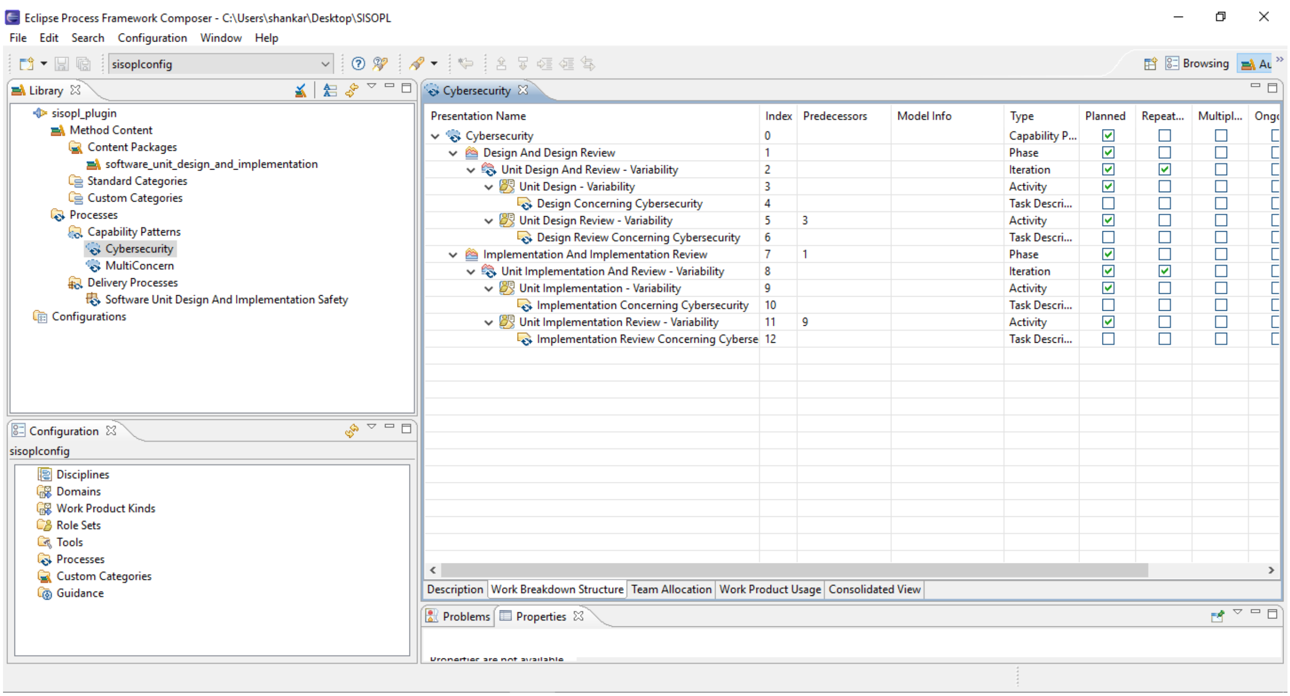
More Information

Guidelines	• Design Notations • Design Principles • Rationale Design Notations • Rationale Design Principles

Figure 34. Task Relationships with Roles, Work Products and Guidance

3.3.1.2. Define Capability Patterns

This corresponds to the Block (B) of the workflow depicted in Figure 25. Capability Patterns reflect best practices which can be further integrated into Delivery Processes. Capability Patterns thus provide a means of rapidly defining Delivery Processes by use of these building blocks as well as capturing the defined best practices. Capability Patterns are similar in structure and definition to Delivery Processes. Capability Patterns for ‘Cybersecurity’ and ‘Multi Concern’ Software Design and Implementation practices are depicted in Figure 35 and Figure 36 respectively.


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL

File Edit Search Configuration Window Help

sisoplconfig

Library
- sisopl_plugin
  - Method Content
    - Content Packages
      - software_unit_design_and_implementation
    - Standard Categories
    - Custom Categories
  - Processes
    - Capability Patterns
      - Cybersecurity
      - MultiConcern
    - Delivery Processes
      - Software Unit Design And Implementation Safety
- Configurations

Configuration

sisoplconfig
- Disciplines
- Domains
- Work Product Kinds
- Role Sets
- Tools
- Processes
- Custom Categories
- Guidance

Cybersecurity

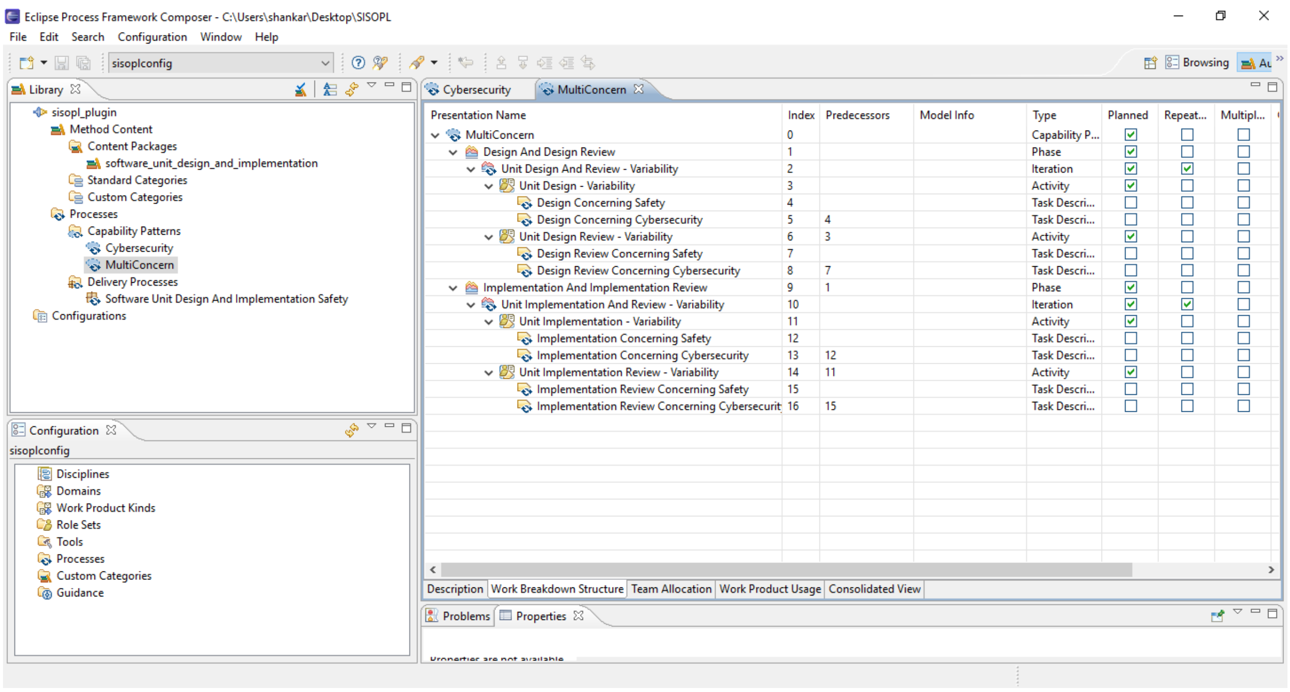
Presentation Name	Index	Predecessors	Model Info	Type	Planned	Repeat...	Multipl...	Ong
Cybersecurity	0			Capability P...	✓	☐	☐	☐
Design And Design Review	1			Phase	✓	☐	☐	☐
Unit Design And Review - Variability	2			Iteration	✓	✓	☐	☐
Unit Design - Variability	3			Activity	✓	☐	☐	☐
Design Concerning Cybersecurity	4			Task Descri...	☐	☐	☐	☐
Unit Design Review - Variability	5	3		Activity	✓	☐	☐	☐
Design Review Concerning Cybersecurity	6			Task Descri...	☐	☐	☐	☐
Implementation And Implementation Review	7	1		Phase	✓	☐	☐	☐
Unit Implementation And Review - Variability	8			Iteration	✓	✓	☐	☐
Unit Implementation - Variability	9			Activity	✓	☐	☐	☐
Implementation Concerning Cybersecurity	10			Task Descri...	☐	☐	☐	☐
Unit Implementation Review - Variability	11	9		Activity	✓	☐	☐	☐
Implementation Review Concerning Cyberse	12			Task Descri...	☐	☐	☐	☐

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

Problems | Properties

Properties are not available

Figure 35. Cybersecurity Capability Pattern


Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL

File Edit Search Configuration Window Help

sisoplconfig

Library
- sisopl_plugin
  - Method Content
    - Content Packages
      - software_unit_design_and_implementation
    - Standard Categories
    - Custom Categories
  - Processes
    - Capability Patterns
      - Cybersecurity
      - MultiConcern
    - Delivery Processes
      - Software Unit Design And Implementation Safety
- Configurations

Configuration

sisoplconfig
- Disciplines
- Domains
- Work Product Kinds
- Role Sets
- Tools
- Processes
- Custom Categories
- Guidance

Cybersecurity | MultiConcern

Presentation Name	Index	Predecessors	Model Info	Type	Planned	Repeat...	Multipl...
MultiConcern	0			Capability P...	✓	☐	☐
Design And Design Review	1			Phase	✓	☐	☐
Unit Design And Review - Variability	2			Iteration	✓	✓	☐
Unit Design - Variability	3			Activity	✓	☐	☐
Design Concerning Safety	4			Task Descri...	☐	☐	☐
Design Concerning Cybersecurity	5	4		Task Descri...	☐	☐	☐
Unit Design Review - Variability	6	3		Activity	✓	☐	☐
Design Review Concerning Safety	7			Task Descri...	☐	☐	☐
Design Review Concerning Cybersecurity	8	7		Task Descri...	☐	☐	☐
Implementation And Implementation Review	9	1		Phase	✓	☐	☐
Unit Implementation And Review - Variability	10			Iteration	✓	✓	☐
Unit Implementation - Variability	11			Activity	✓	☐	☐
Implementation Concerning Safety	12			Task Descri...	☐	☐	☐
Implementation Concerning Cybersecurity	13	12		Task Descri...	☐	☐	☐
Unit Implementation Review - Variability	14	11		Activity	✓	☐	☐
Implementation Review Concerning Safety	15			Task Descri...	☐	☐	☐
Implementation Review Concerning Cybersecurit	16	15		Task Descri...	☐	☐	☐

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

Problems | Properties

Properties are not available

Figure 36. Multi Concern Capability Pattern

3.3.1.3. Define Delivery Processes

This corresponds to the Block (C) of the workflow depicted in Figure 25. Processes are defined to show how work is performed as part of a development cycle and also help in defining project milestones and how they can be achieved. Processes may be either ordered sequentially or be semi-ordered as iterations of work as appropriate to the development process employed. Further, the tasks can be modelled iteratively while defining processes to factor the impact of a task on the behaviour of a previous task and vice versa. These iterations can be carried on until you reach a design where such impact is absent or acceptable.

To illustrate the definition of a delivery process in the context of process-related dependability co-assessment, previous work is reused. In particular, the work presented in [56] is reused. In this work, a co-analysis and co-engineering approach to linking safety and security architectural patterns, called 'Pattern Engineering Lifecycle' (depicted in Figure 37), is described, within the context of ISO 26262 (automotive domain specific safety standard) and SAE J3061 (cybersecurity process framework).

In [56], no process-related pattern (no capability pattern) is detailed. Authors only explain how the abstract process given in Figure 37 could be instantiated via some more detailed process steps which include the exploitation of architectural patterns.

Here, instead, the focus is on process-related patterns and on how the abstract process given in Figure 37 could be interpreted and modelled in a more refined way in EPF Composer. A possible interpretation would be that each engineering block is modelled into a process-based pattern (capability pattern in EPF Composer).

Since, however, in what follows the goal is to illustrate a re-configuration of a safety-related process model, the safety-related engineering block is simply modelled as an instantiation of a safety-related capability pattern.

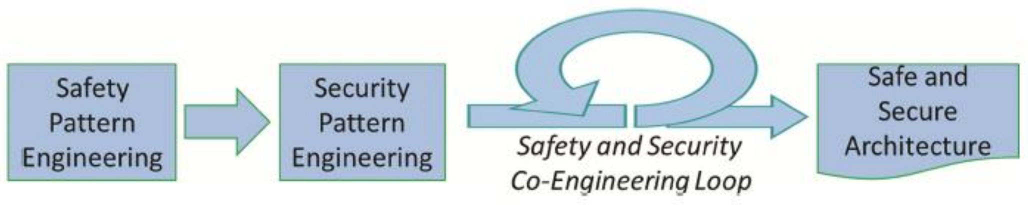

Safety Pattern Engineering
Security Pattern Engineering
Safety and Security Co-Engineering Loop
Safe and Secure Architecture

Figure 37. Pattern Engineering Lifecycle [56]

The Security Pattern Engineering block shown in Figure 37 corresponds to the Capability Pattern for Cybersecurity depicted in Figure 35. In the example illustrated in Figure 39, we instantiate the Delivery Process for Safety (note that also for the Safety Pattern Engineering block, a Capability Pattern could have been used).

In Figure 36, the Capability pattern encompassing both Safety Pattern Engineering and Security Pattern Engineering is depicted. An interpretation of the 'Pattern Engineering Lifecycle' is depicted in Figure 38. Figure 38 depicts the iterations to factor the impact of safety and cybersecurity requirements on design. Similar iterations may be performed for implementation also.

The blocks Design Concerning Safety and Review Concerning Safety (in Figure 38) correspond to the block Safety Pattern Engineering (in Figure 37) while the blocks Design Concerning Cybersecurity and Review Concerning Cybersecurity (in Figure 38) correspond to the block Security Pattern Engineering (in Figure 37). Also, the blocks Safety Pattern Engineering and Security Pattern Engineering (in Figure 37) correspond to the Multi Concern Capability Pattern depicted in Figure 36. The Safety and Security Co-Engineering Loop (in Figure 37) is represented by the decisions boxes in Figure 38 and realized by the iterations defined in Figure 36 (Unit Design And Review – Variability and Unit Implementation And Review – Variability).

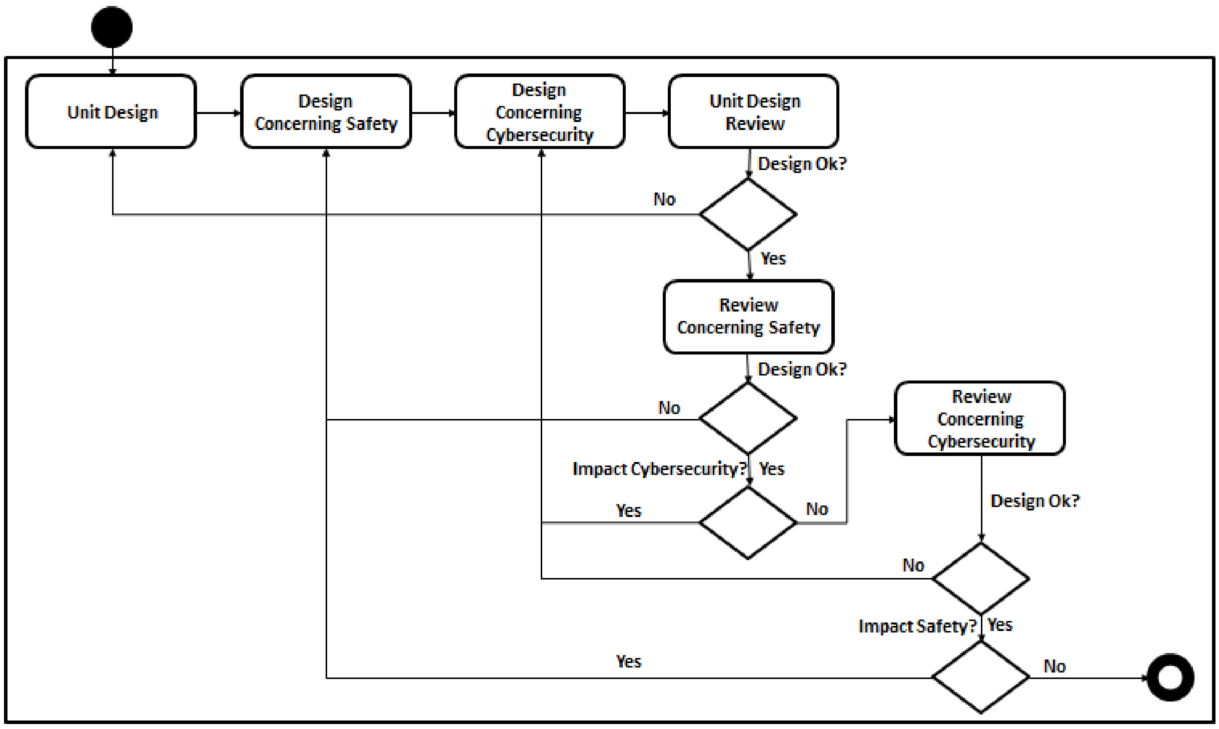
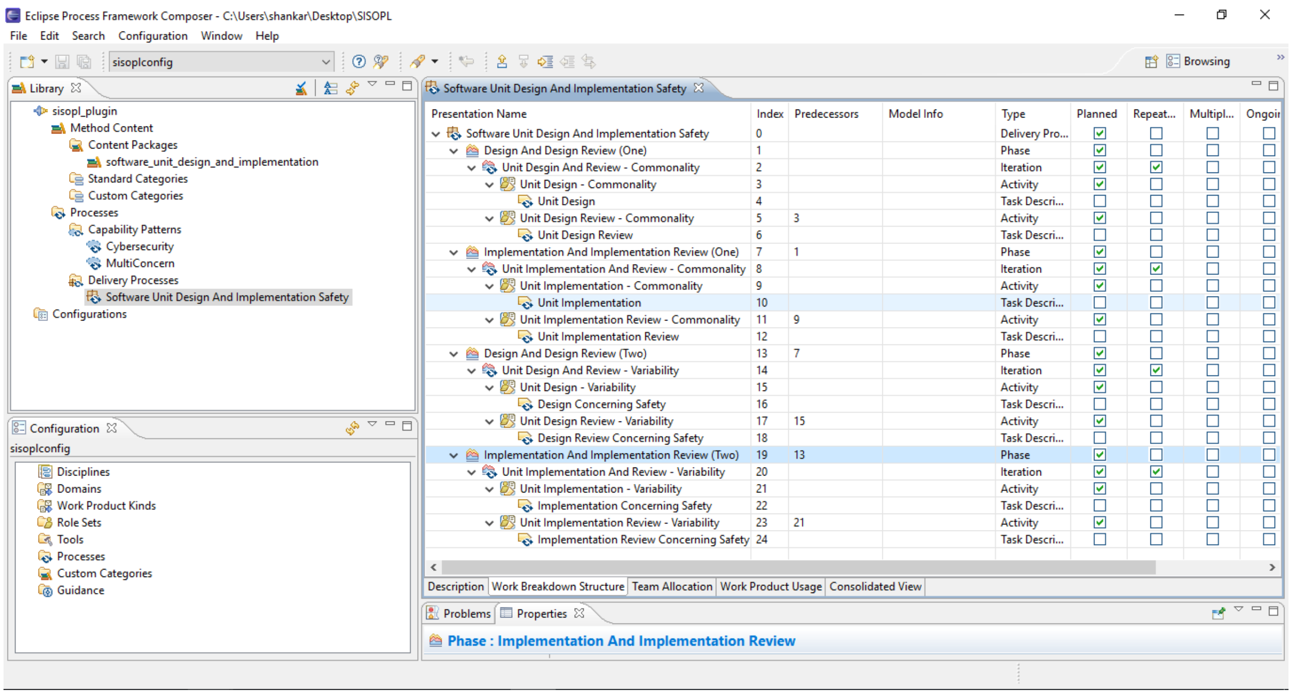


Figure 38. Iterative Design Process Factoring Safety and Cybersecurity Requirements

Processes may be expressed as work breakdown structures or workflows. Figure 39 depicts a fragment of the safety design process (Software Unit Design and Implementation) as a work breakdown structure organized into phases, iterations, activities and tasks as well as their precedence relations. Also, role and work product associations to activities are specified in the work breakdown structure. Further, the roles, work products and task steps can be modified (addition, suppression, resequencing) to match the frame of reference of the process being defined. The 'Software Unit Design and Implementation' (SUDI) process can be defined as a Capability Pattern which may be further reused to define either a safety-based design process, a cybersecurity based design process or a combined design process. The SUDI process (Safety) is composed of two phases, namely, 'Design and Design Review' and 'Implementation and Implementation Review' for commonality and two similar phases for variability. Each phase is broken down into iterations and the activities and the tasks to be repeated in the iterations.



Eclipse Process Framework Composer - C:\Users\shankar\Desktop\SISOPL

Library
- sisopl_plugin
  - Method Content
    - Content Packages
      - software_unit_design_and_implementation
    - Standard Categories
    - Custom Categories
  - Processes
    - Capability Patterns
      - Cybersecurity
      - MultiConcern
    - Delivery Processes
      - Software Unit Design And Implementation Safety
- Configurations

Configuration: sisoplconfig
- Disciplines
- Domains
- Work Product Kinds
- Role Sets
- Tools
- Processes
- Custom Categories
- Guidance

Software Unit Design And Implementation Safety

Presentation Name	Index	Predecessors	Model Info	Type	Planned	Repeat...	Multipl...	Ongoir
Software Unit Design And Implementation Safety	0			Delivery Pro...	✓	☐	☐	☐
Design And Design Review (One)	1			Phase	✓	☐	☐	☐
Unit Desgin And Review - Commonality	2			Iteration	✓	✓	☐	☐
Unit Design - Commonality	3			Activity	✓	☐	☐	☐
Unit Design	4			Task Descri...	☐	☐	☐	☐
Unit Design Review - Commonality	5	3		Activity	✓	☐	☐	☐
Unit Design Review	6			Task Descri...	☐	☐	☐	☐
Implementation And Implementation Review (One)	7	1		Phase	✓	☐	☐	☐
Unit Implementation And Review - Commonality	8			Iteration	✓	✓	☐	☐
Unit Implementation - Commonality	9			Activity	✓	☐	☐	☐
Unit Implementation	10			Task Descri...	☐	☐	☐	☐
Unit Implementation Review - Commonality	11	9		Activity	✓	☐	☐	☐
Unit Implementation Review	12			Task Descri...	☐	☐	☐	☐
Design And Design Review (Two)	13	7		Phase	✓	☐	☐	☐
Unit Design And Review - Variability	14			Iteration	✓	✓	☐	☐
Unit Design - Variability	15			Activity	✓	☐	☐	☐
Design Concerning Safety	16			Task Descri...	☐	☐	☐	☐
Unit Design Review - Variability	17	15		Activity	✓	☐	☐	☐
Design Review Concerning Safety	18			Task Descri...	☐	☐	☐	☐
Implementation And Implementation Review (Two)	19	13		Phase	✓	☐	☐	☐
Unit Implementation And Review - Variability	20			Iteration	✓	✓	☐	☐
Unit Implementation - Variability	21			Activity	✓	☐	☐	☐
Implementation Concerning Safety	22			Task Descri...	☐	☐	☐	☐
Unit Implementation Review - Variability	23	21		Activity	✓	☐	☐	☐
Implementation Review Concerning Safety	24			Task Descri...	☐	☐	☐	☐

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

Phase : Implementation And Implementation Review

Figure 39. Software Unit Design and Implementation Delivery Process

Figure 40 depicts the Process Diagram of the delivery process while Figure 41 depicts the Detailed Activity Diagram. The Process Diagram is at the phase level while the Detailed Activity Diagram is at an activity level. The Detailed Activity Diagram shows the tasks which make up the activity, the associated roles and work products.

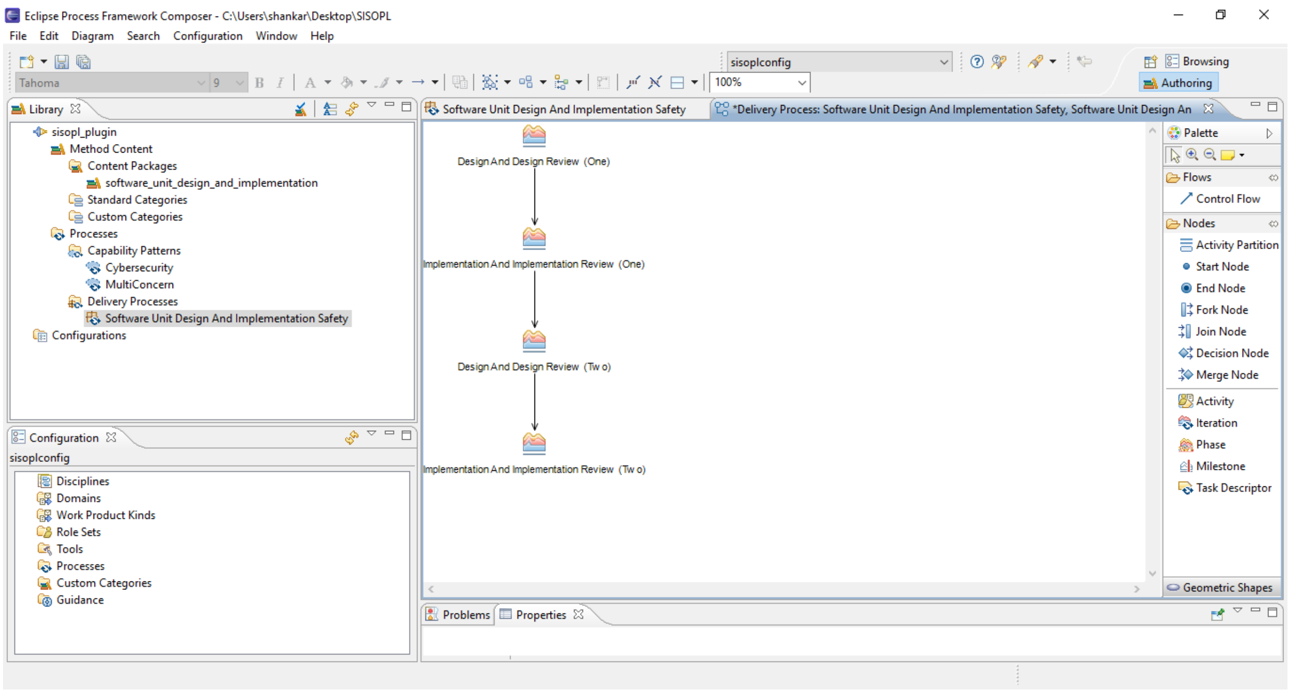


Figure 40. Process Diagram - Software Unit Design and Implementation Delivery Process

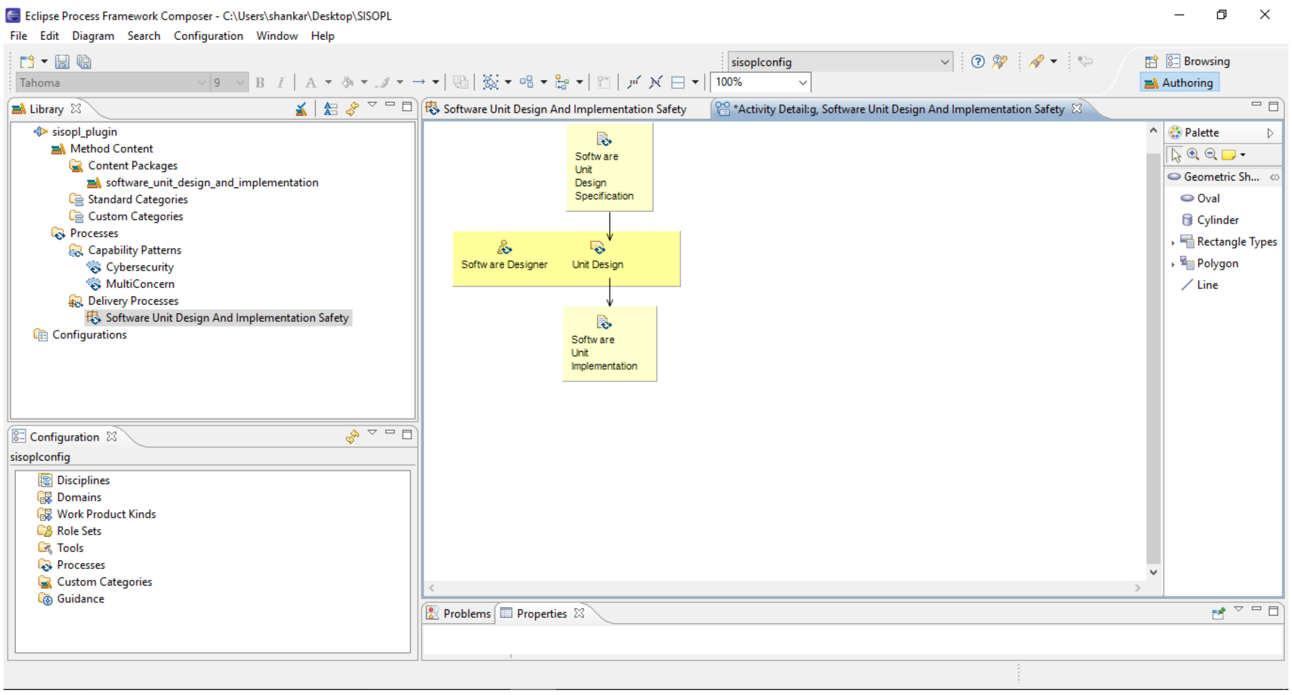
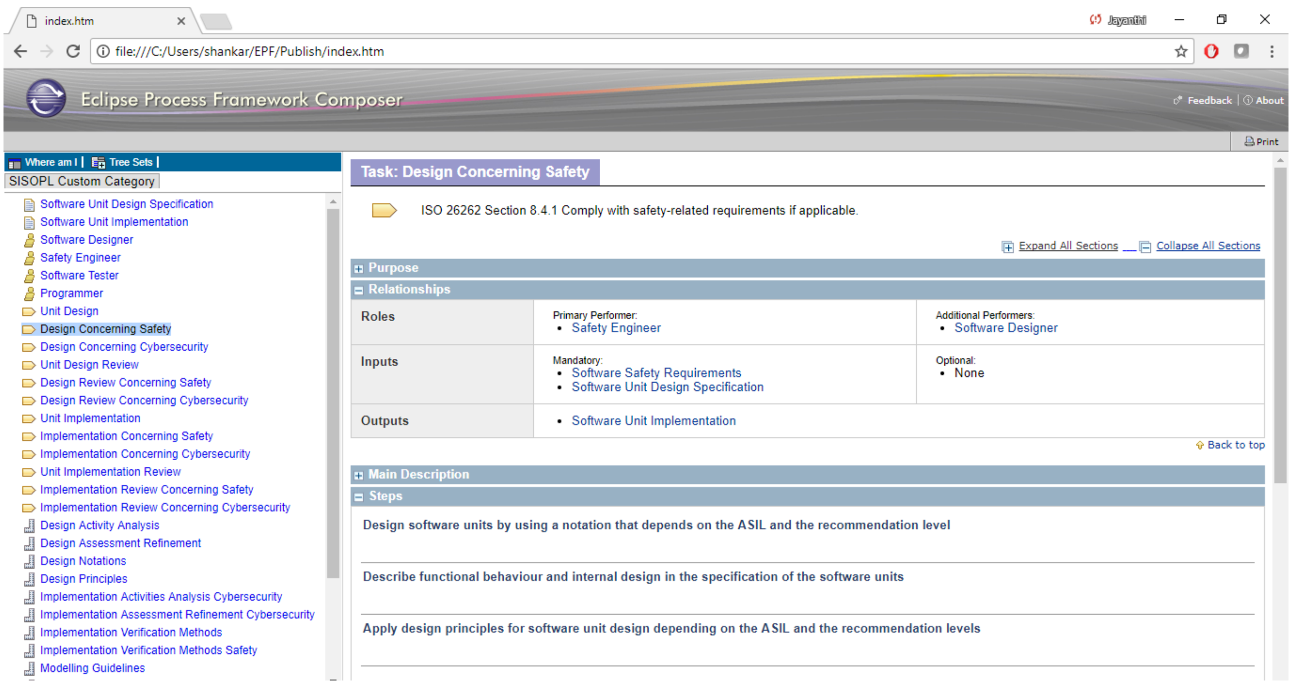


Figure 41. Detailed Activity Diagram – Software Unit Implementation

3.3.1.4. Publish Method Configuration

This corresponds to the Block (D) of the workflow depicted in Figure 25. Publishing Method Configuration enables sharing of method content, guidance and processes with members of the Project Team. Publishing is a two-stage activity consisting of defining a Method Configuration made up of a selection of method content and processes from one or more plugins followed by publishing it as a website. The contents of the published website, allow navigation between the various elements contained, by way of hyperlinks. A fragment of the Published Method Content is depicted in Figure 42.


index.htm
file:///C:/Users/shankar/EPF/Publish/index.htm
Eclipse Process Framework Composer
Feedback | About
Print
Where am I | Tree Sets |

SISOPL Custom Category

- Software Unit Design Specification
- Software Unit Implementation
- Software Designer
- Safety Engineer
- Software Tester
- Programmer
- Unit Design
- Design Concerning Safety
- Design Concerning Cybersecurity
- Unit Design Review
- Design Review Concerning Safety
- Design Review Concerning Cybersecurity
- Unit Implementation
- Implementation Concerning Safety
- Implementation Concerning Cybersecurity
- Unit Implementation Review
- Implementation Review Concerning Safety
- Implementation Review Concerning Cybersecurity
- Design Activity Analysis
- Design Assessment Refinement
- Design Notations
- Design Principles
- Implementation Activities Analysis Cybersecurity
- Implementation Assessment Refinement Cybersecurity
- Implementation Verification Methods
- Implementation Verification Methods Safety
- Modelling Guidelines

Task: Design Concerning Safety

ISO 26262 Section 8.4.1 Comply with safety-related requirements if applicable.

Expand All Sections __ Collapse All Sections

Purpose

Relationships

Roles	Primary Performer: • Safety Engineer	Additional Performers: • Software Designer
Inputs	Mandatory: • Software Safety Requirements • Software Unit Design Specification	Optional: • None
Outputs	• Software Unit Implementation	

Back to top

Main Description

Steps

Design software units by using a notation that depends on the ASIL and the recommendation level

Describe functional behaviour and internal design in the specification of the software units

Apply design principles for software unit design depending on the ASIL and the recommendation levels

Figure 42. Fragment of Published Method Content

3.3.2. BVR Workflow

In this subsection, the BVR Workflow is explained. This workflow exploits the process model modelled by applying the EPF Composer-related workflow (left-hand side sub-workflow in Figure 25).

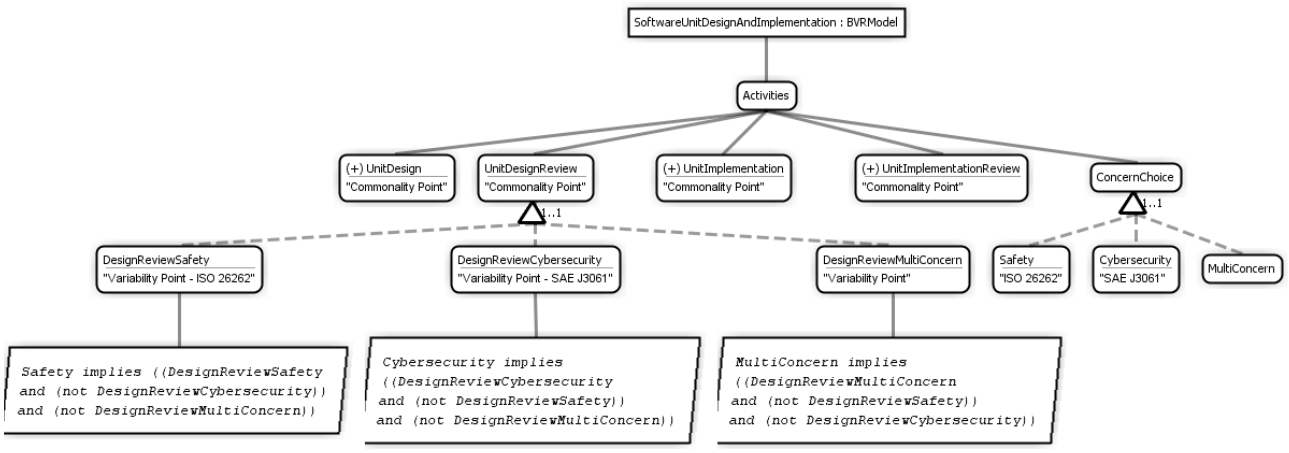
3.3.2.1. Create Variability Model

This corresponds to the Block (E) of the workflow depicted in Figure 25. The Variability Model is created using the VSpec editor of the BVR tool thus enabling modelling the features of the software development process in the form of a feature diagram.

The VSpec editor also provides means to specify multiplicity (concepts such as exclusive-or, one of, etc.) and constraints. Constraints represent cross-feature dependencies. A feature may require the presence or absence of another feature.

The Software Unit Design and Implementation variability models for Safety, Cybersecurity and Multi Concern are created using the VSpec editor and are derived essentially from the base model (with the exception of constraints) described in Subsection 3.3.1. A fragment of the feature diagram using the BVR VSpec editor is depicted in Figure 43. Note that the '+' symbol indicates that the VSpec Model displays a feature, which has been minimised.

As it can be seen in Figure 43, the feature ConcernChoice takes one of three values, namely Safety, Cybersecurity and MultiConcern, which are depicted as an exclusive-or feature. The constraint is specified in the parallelogram and is composed of features connected by operators 'and', 'not', 'implies' and parentheses. Optional features, as determined by the specified constraints, are connected with dotted lines while mandatory features are connected with solid lines. The Variability Model use case is described in Subsection 4.3.8.


SoftwareUnitDesignAndImplementation : BVRModel
Activities
(+) UnitDesign "Commonality Point"
UnitDesignReview "Commonality Point"
(+) UnitImplementation "Commonality Point"
(+) UnitImplementationReview "Commonality Point"
ConcernChoice
1..1
1..1
DesignReviewSafety "Variability Point - ISO 26262"
DesignReviewCybersecurity "Variability Point - SAE J3061"
DesignReviewMultiConcern "Variability Point"
Safety "ISO 26262"
Cybersecurity "SAE J3061"
MultiConcern

```
Safety implies ((DesignReviewSafety
and (not DesignReviewCybersecurity))
and (not DesignReviewMultiConcern))
```

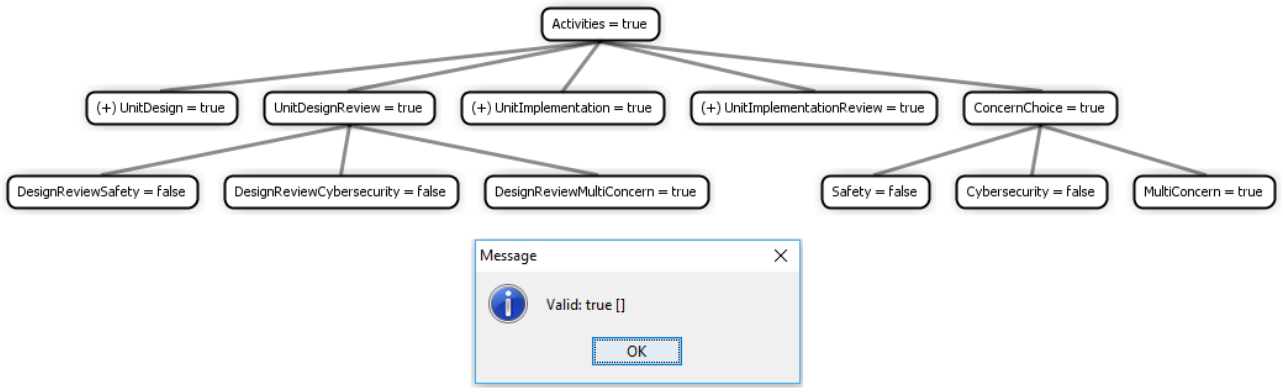
```
Cybersecurity implies
((DesignReviewCybersecurity
and (not DesignReviewSafety))
and (not DesignReviewMultiConcern))
```

```
MultiConcern implies
((DesignReviewMultiConcern
and (not DesignReviewSafety))
and (not DesignReviewCybersecurity))
```

Figure 43. BVR Feature Model using VSpec Editor

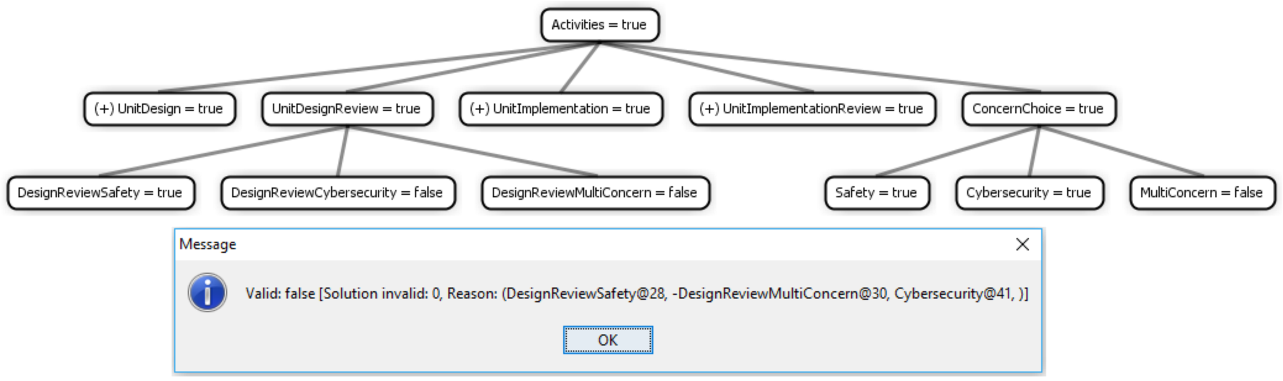
3.3.2.2. Resolve Configuration

This corresponds to the Block (F) of the workflow depicted in Figure 25. The resolution results in the generation of the Resolution model from the Variability model which was created using the VSpec editor. This is performed using the Resolution editor. The Variability Constraints represent valid resolutions of the model. The resolution model also looks very similar to the variability model created earlier and resembles a tree structure. The Resolution editor allows for validation of the resolved model allowing the ability to resolve the model correctly based on the specified constraints and cardinality. Figure 44 and Figure 45 depict cases of valid resolution and invalid resolution respectively.


Activities = true
(+) UnitDesign = true
UnitDesignReview = true
(+) UnitImplementation = true
(+) UnitImplementationReview = true
ConcernChoice = true
DesignReviewSafety = false
DesignReviewCybersecurity = false
DesignReviewMultiConcern = true
Safety = false
Cybersecurity = false
MultiConcern = true
Message
OK

Valid: true []

Figure 44. BVR Resolution Model with Valid Resolution


Activities = true
(+) UnitDesign = true
UnitDesignReview = true
(+) UnitImplementation = true
(+) UnitImplementationReview = true
ConcernChoice = true
DesignReviewSafety = true
DesignReviewCybersecurity = false
DesignReviewMultiConcern = false
Safety = true
Cybersecurity = true
MultiConcern = false
Message
OK

Valid: false [Solution invalid: 0, Reason: (DesignReviewSafety@28, -DesignReviewMultiConcern@30, Cybersecurity@41,)]

Figure 45. BVR Resolution Model with Invalid Resolution

In Figure 45, we can see that both ‘Safety’ and ‘Cybersecurity’ choices are True in an exclusive-or relationship, thereby resulting in an invalid resolution of the model. Resolution of all variabilities correctly is a prerequisite for proceeding to the product realization steps, namely ‘Define Mappings’ and ‘Generate Process Model’.

3.3.2.3. Realise Model

This corresponds to the Block (G) of the workflow depicted in Figure 25. The Realisation Model is created by using the Realization Editor. During the editing of the Realization Model, the Resolution Model and the Base Model are used. In particular, the Resolution Model indicates which features should be part of the Realization Model. The Realization Model is obtained by modifying the Base Model through a series of transformations, which apply fragment substitutions (consisting of Placements and Replacements). Once the Realisation Model is created, it can be exported back to the Base Model Editor (i. e., EPF Composer).

For instance, Software Unit Design and Implementation Safety Delivery Process, depicted in Figure 39, can be used as Base Model to create a new process model representing a multi-concern process. A complete illustration regarding the creation of a realisation model is given in Chapter 4.

3.4. Standard-related Dependability Co-assessment via OpenCert Workflow (*)

This section is connected with the previous one, but here the focus is on the standard-related dependability co-assessment.

Standardization for safety and security is still separate. In AMASS we have tried to show this separation using OpenCert, where the different standards are modelled as different reference frameworks, however we support the specification of equivalences between the standards. An expert in both standards, or different experts, each with expertise in a single standard, working together, must map concepts (activities, artefacts, and requirements) from source and target standards by using OpenCert Equivalence Maps. As a result of this activity, a model of Equivalence Maps between source and target Reference Frameworks will be generated. More specifically, its workflow is depicted in Figure 46. This workflow describes the work that should be conducted by using OpenCert. The complete process is described in deliverable D6.8 [8] as “Cross-Standard reuse”.

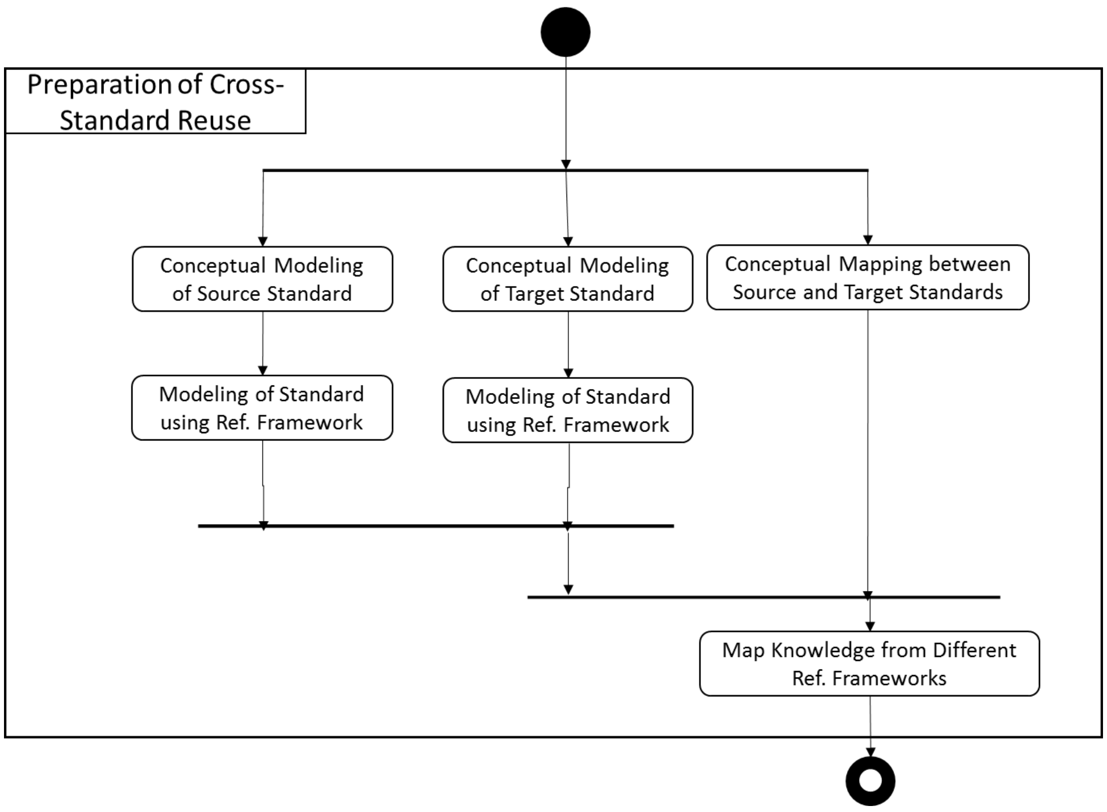

Preparation of Cross-Standard Reuse
Conceptual Modeling of Source Standard
Conceptual Modeling of Target Standard
Conceptual Mapping between Source and Target Standards
Modeling of Standard using Ref. Framework
Modeling of Standard using Ref. Framework
Map Knowledge from Different Ref. Frameworks

Figure 46. Sub-activities related to the Preparation of Cross-Standard Reuse

3.5. System Dependability Co-Analysis (*)

In this section, the workflows for guiding users interested in performing system dependability co-analysis are given. More specifically, three different techniques can be used.

3.5.1. System Dependability Co-Analysis via Papyrus SSE

To address multi-concern assurance, Papyrus SSE strongly relies upon the principle of elements reuse. In particular, the reuse of modelling artefacts is a core technique of the approach. As it is seen in Figure 47, a model of the target system is first designed by the user. To do so, several standardised languages are supported like UML, SysML, BPMN and RobotML [49]. Languages specific to an engineering domain can be specified and implemented in Papyrus SSE, if they are based on the Core supported languages, as for example CHESSML.

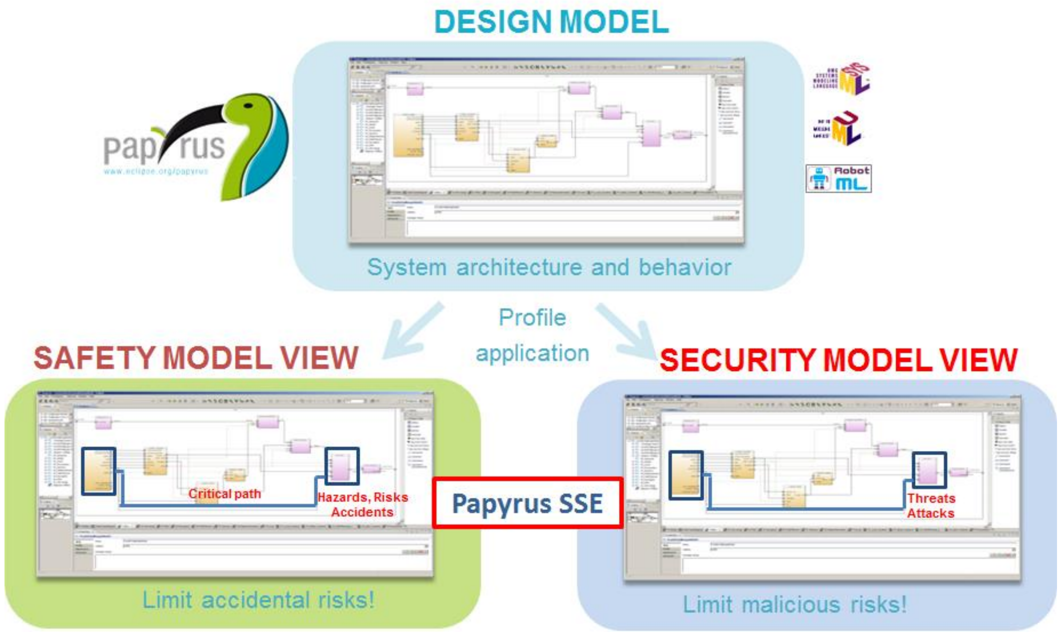


Figure 47. Annotation of the system model to conduct safety and/or security analyses

Once a first version of the system model is complete, the user can select to conduct safety and/or security analyses. To do so, the user should apply the dedicated profile(s) which, among others, allow to annotate the system model by adding the elements required by the analysis: the annotations introduce functional and non-functional attributes to be addressed and/or evaluated. The annotations finally extend and produce safety and security-oriented models. The annotated models and the analyses outcomes can always be traced from their sources, including the requirements they fulfil. Such traceability plays a key role for identifying commonalities between analyses (and respective modelling elements), which is necessary to support a joint safety-security analysis. In the following subsections, we illustrate the main tasks to be executed by users in order to apply the approach.

3.5.1.1. Papyrus for Safety

Papyrus for Safety (Papyrus4Safety) denotes the platform used to analyse safety aspects. In Figure 48, we illustrate the lifecycle that is supported by the tool to conduct safety-oriented analyses (HARA, FMEA, FTA, etc.). Each analysis demands the application of a profile to be applied on the system model. The profile stereotypes can be managed either manually or automatically. In the first case, the user creates basic elements, apply the needed stereotype and fill its attributes. In the second case, dedicated functions are available via pop-up menus which can be executed targeting all elements in the model.

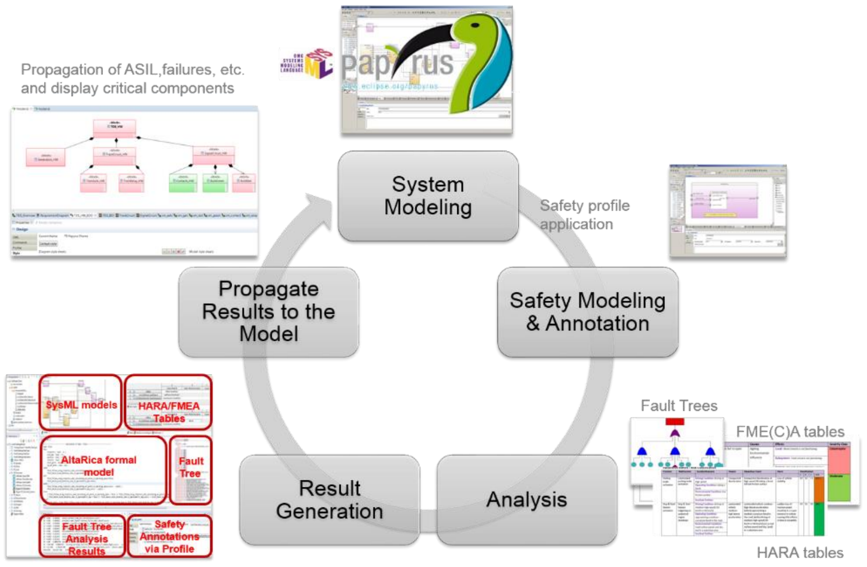

Propagation of ASIL,failures, etc. and display critical components
System Modeling
Safety profile application
Safety Modeling & Annotation
Analysis
Result Generation
Propagate Results to the Model
SysML models
HARA/FMEA Tables
AltaRica formal model
Fault Tree
Fault Tree Analysis Results
Safety Annotations via Profile
Fault Trees
FME(C)A tables
HARA tables

Figure 48. Lifecycle supported by Papyrus4Safety for model-based safety analysis

3.5.1.2. Papyrus for Security

Papyrus for Security (Papyrus4Security) denotes the platform used to analyse security aspects. In Figure 49, we illustrate the main phases supported by the framework. The phases correspond to the workflow a user can follow in order to cover a full cycle of the risks assessment and the system securing processes: 1) definition of the perimeter of the analysis and security metrics (criteria and scales) parameterisation; 2) primary assets, and security goals definition; 3) supporting assets analysis, vulnerabilities identification and security countermeasures definition; 4) threats scenarios, propagation analysis and impact evaluation; 5) attacks trees definition and likelihood evaluation: and 6) security risk assessment and reduction. The details about the different analyses are already explained in D3.8 [7] and will not be repeated here.

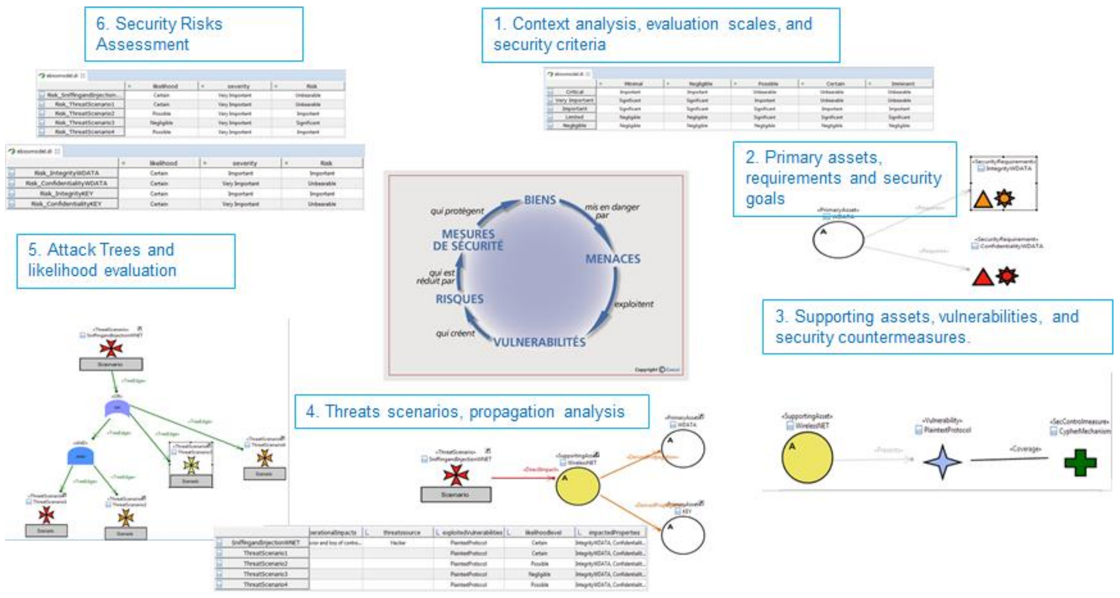

6. Security Risks Assessment
1. Context analysis, evaluation scales, and security criteria
2. Primary assets, requirements and security goals
3. Supporting assets, vulnerabilities, and security countermeasures.
4. Threats scenarios, propagation analysis
5. Attack Trees and likelihood evaluation
BIENS
MENACES
VULNERABILITÉS
RISQUES
MESURES DE SÉCURITÉ
qui protègent
mis en danger par
exploitent
qui créent
qui est réduit par

Figure 49. Main phases supported by Papyrus4Security

3.5.1.3. Exploiting Papyrus SSE mono-concern results for multi-concern perspective

Papyrus for Safety and Security Engineering (Papyrus SSE) targets three major milestones:

- Support for safety standalone engineering

- Support for security standalone engineering
- Support for safety-security co-engineering

For now, the safety and security standalone functionalities of Papyrus SSE are the most developed and mature. The safety-security co-engineering has emerged in recent years as a viable and promising approach that aims to identify and exploit commonalities and dependencies between analyses from early phases of design. As part of this multi-concern perspective, the CEA team is currently developing Papyrus extensions to integrate multi-concern aspects. To do so, several standards are being taken as reference. Among them, we can mention the following:

- EUROCAE ED-202 [46] and ED-203 [47], for the airworthiness security process and methods, respectively.
- DO-356 [48], airworthiness considerations and methods.

Of course, these standards are oriented to the aeronautics domain, however, they provide important insights on the processes and methods for safety-security co-engineering for any critical domain. On one side, the aeronautics domain is clearly safety critical. On the other side, the increasing connectivity of airplane on-board systems (civil and military) is motivating the involved sectors to initiate discussions on the potential impact of security threats. Indeed, as in other systems (e.g., Industrial Control Systems), the todays aeronautics systems were mostly designed without security in mind and their increasing connectivity imposes several risks that need to be evaluated. In this context, the road map of Papyrus SSE extensions already considers the next aspects for multi-concern assurance:

- **Commonalities:** the commonalities between safety and security analyses need to be thoroughly considered. On one side, the syntactical similarity of fundamental notions is one of the major stakes. It has been identified as necessary to regroup the concepts that are syntactically similar; however, in the end, it is not sufficient. Indeed, a major challenge emerge when considering the semantical differences of syntactically similar concepts. For instance, the notion of feared events exists in both safety and security analyses. However, the nature, techniques and conceptual elements involved in feared events elicitation may considerably differ for safety and security analyses and for specific study cases. It is becoming clear that a one-to-one security-safety concepts mapping is not any more a target but a possible reference to construct.
- **Particularities:** as a consequence of the drawbacks already identified for settling a full and consistent match between safety and security aspects, concepts and methods, the identification of safety and security particularities and their co-existence arise, i.e., multi-concern co-engineering. Regarding Papyrus for SSE, the meta-models and profiles already implemented to support standalone security and safety analyses are the basis to construct and specify not only commonalities but also their particularities. Thus, for instance, the notions of security criteria (also named security attributes) namely confidentiality, authenticity, privacy, etc. (which do not have a safety-oriented counterpart) will be part of the security approach particularities. On the other side, the notions of accidental failure, failure rate, failure propagation, etc., will be part of the safety approach particularities.
- **Evaluation metrics:** safety and security methods already introduce metrics to evaluate risks. However, the nature of safety and security events is not necessarily the same. Even if statistical and probabilistic methods can be applied to evaluate both safety and security events occurrence, the first ones mostly obey to the physical nature of the components and their exposition to operational conditions whereas the second ones mostly obey to the motivations, gains and rationales of human beings. Regarding Papyrus for SSE, a perimeter is settled to separate common from specific aspects. By doing so, it is ensured that the evaluation metrics, which can be independently used for standalone safety or security risks evaluation, truly converge over common spaces.
- **Safety-security techniques integration:** the identification of commonalities and particularities, and the – well – defined evaluation metrics are the basis upon which the safety and security-oriented techniques can be integrated. In particular, the integration of Fault Trees and Attack Trees

techniques is a promissory but challenging task targeted in the Papyrus SSE roadmap. Indeed, even if there exist several approaches that show the feasibility of such integration, several issues still need to be addressed, e.g., language, formalization, semantics, etc. Even so, several aspects are currently inspected, implemented and evaluated. In particular, a consistent hybrid Fault-Attack-Defence Tree can be constructed upon the notion of safety-oriented attack and non-safety-oriented attack. To accomplish a consistent integration, several issues still need to be solved:

  - *Abstraction levels heterogeneity:* the safety and security parts of a hybrid tree may need to be imported from models developed as standalone entities. The different levels of abstraction used for modelling need to be considered and harmonized. This becomes evident when targeting the integration of knowledge bases, e.g., attacks, vulnerabilities, attack scenarios, etc.
  - *Safety and security events complexity:* as previously mentioned, the nature of safety and security events is not necessarily the same. For instance, the propagation of a failure is determined by a cause-effect rule that is often written as a Boolean formula. However, the propagation of an attack is often based upon the notion of attack action or step. Whereas the propagation of a failure obeys the cause-effect rule involving binary inputs and outputs, the propagation of an attack action is far more complex since it may involve other elements like vulnerabilities, countermeasures, attacker resources, skills and motivations, window of opportunity, etc. The sound evaluation of propagations complexity needs to be thoroughly considered.

- **Safety-security border:** as long as the Papyrus SSE extensions are developed and tested on use cases, it becomes clear that a logical border between safety and security spaces may soon appear. Indeed, for some cases, the security analysis may be limited to cover only safety-oriented impacts. In those cases, the targets of an attack are inferred (by considering the safety impacts) and a border between attack vectors and failures propagations can be settled. For other cases, a safety analysis may be extended or completed by a security one. When this happens, first the potential attacks are elicited and then cross-related to the safety analysis; in addition, a logical border between attacks and safety analysis can also be identified and settled. The cases in which neither priority nor sequencing exist between safety and security analyses are rare. That is why, the identification of a logical border between safety and security spaces can be foreseen. If present, such a border can be exploited when addressing the particularities of safety and security analyses.

3.5.2. System Dependability Co-Analysis via Safety Architect

The methodological guide proposed for using Safety Architect [14] and Cyber Architect [15] (presented briefly in Section 2.3.6) is based on separation of concerns and co-engineering approaches. The main motivation for separation of concerns is that every engineer (be it an architect, a security or a safety engineer) can focus on his/her concern solely because certain domains, such as safety and security domains, are quite different in terms of practices and concepts used. The main motivation for co-engineering is that today no single modelling language and tool can cover all the system engineering activities (specification, analysis, design, verification and validation) and multi-concerns (e.g., Safety and Safety co-analysis and co-validation).

The methodology is based on the seamless interoperability between AMASS platform (CHESS tool and OpenCert) and ALL4TEC's tools as presented in Figure 50.

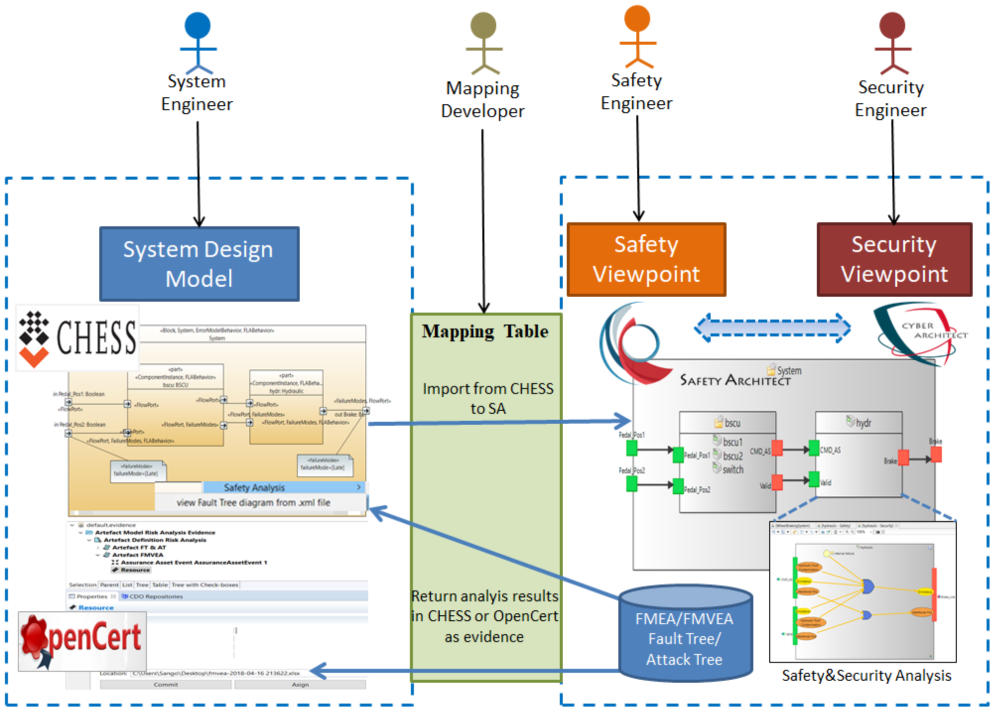


Figure 50. Interoperability between AMASS platform (CHESS and OpenCert) with Safety/Cyber Architect tools

A possible usage scenario is composed by the following steps:

- **Step 1:** System architecture model in CHESS tool.
- **Step 2:** Export/Import system architecture model from CHESS to Safety Architect for safety analysis.
- **Step 3:** Cyber-security analysis in Cyber Architect.
- **Step 4:** Import of security analysis artefacts from Cyber Architect in Safety Architect for co-analysis.
- **Step 5:** Safety & Security co-analysis in Safety Architect.
- **Step 6:** Generation of Safety & Security propagation trees for manual trade-offs between safety and security engineers.
- **Step 7:** Visualisation of propagation trees in the AMASS Platform (CHESS) to facilitate the co-engineering between Safety & Security Engineer and System Architect in order to facilitate the re-design of system architecture if needed.
- **Step 8:** Use of Safety & Security propagation trees as Safety & Security evidences in AMASS platform (OpenCert).

A workflow including the previously listed steps is given in Figure 51.

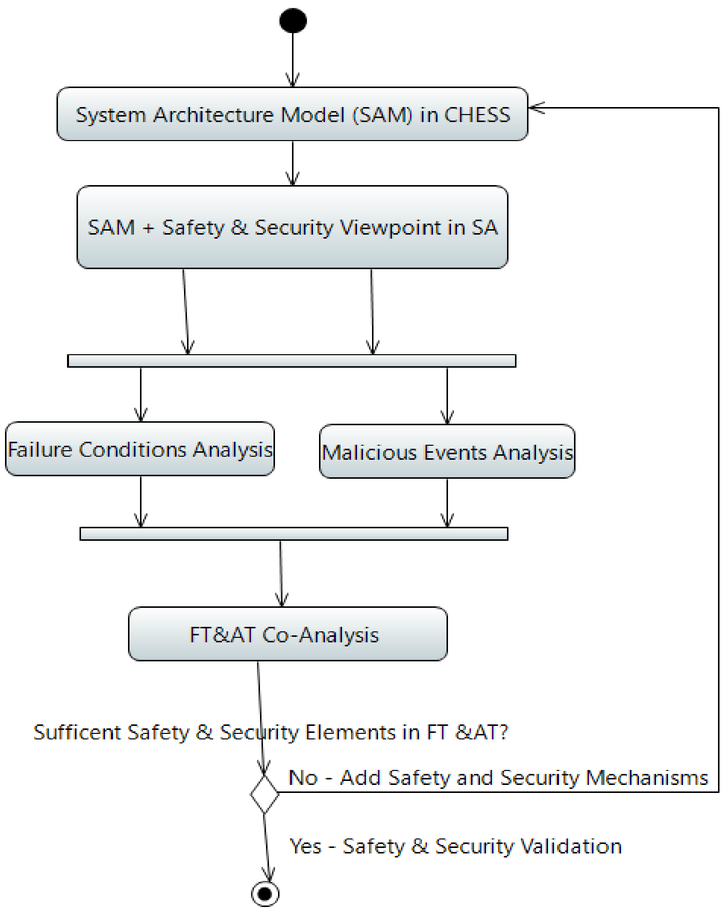

System Architecture Model (SAM) in CHESS
SAM + Safety & Security Viewpoint in SA
Failure Conditions Analysis
Malicious Events Analysis
FT&AT Co-Analysis
Sufficent Safety & Security Elements in FT &AT?
No - Add Safety and Security Mechanisms
Yes - Safety & Security Validation

Figure 51. Workflow regarding system dependability co-analysis via Safety Architect

3.5.3. System Dependability Co-Analysis via ConcertoFLA

In this subsection, ConcertoFLA is proposed as a means for performing system dependability co-analysis. More specifically, it is proposed that the Safety engineer models the safety-related threats (e.g., fault, error, failure) and their propagation and that the Security engineer models the security-related threats (e.g., attack, vulnerability and threat) and related propagation. Once this is done, it is proposed to apply ConcertoFLA dependability co-analysis and automatically generate a multi-concern fault tree using the results of this co-analysis. In addition to the generation of certifiable evidence for assurance purposes, the analysis results also contribute in the evaluation of the trade-offs between the multi-concerns. In what follows, the general workflow for addressing system dependability co-analysis via ConcertoFLA is given and then applied on a simple example.

3.5.3.1. Workflow

This subsubsection describes the workflow for defining a system and performing dependability analysis for assuring different non-functional properties of the system. The AMASS platform, via inclusion of CHESS toolset, enables the support for system design, dependability modelling and analysis.

The activity diagram shown in Figure 52, illustrates the steps for system design and co-analysis via ConcertoFLA. The initial step is to define the system by modelling its components and the interactions. The next step is to model the failure behaviour for all components. After that, the failure behaviour of the components is specialized to address the security concern for these components. Then, the ConcertoFLA dependability co-analysis is performed on the system design to identify the system behaviour in the presence of faults and security attacks. In the next step, the ConcertoFLA co-analysis results are transformed to generate the multi-concern fault tree. Based on this, a decision is made for introducing the robustness, safety and/or security measures by refactoring the system. This process is repeated iteratively, until the sufficient level of these concerns (safety and security) is not met.

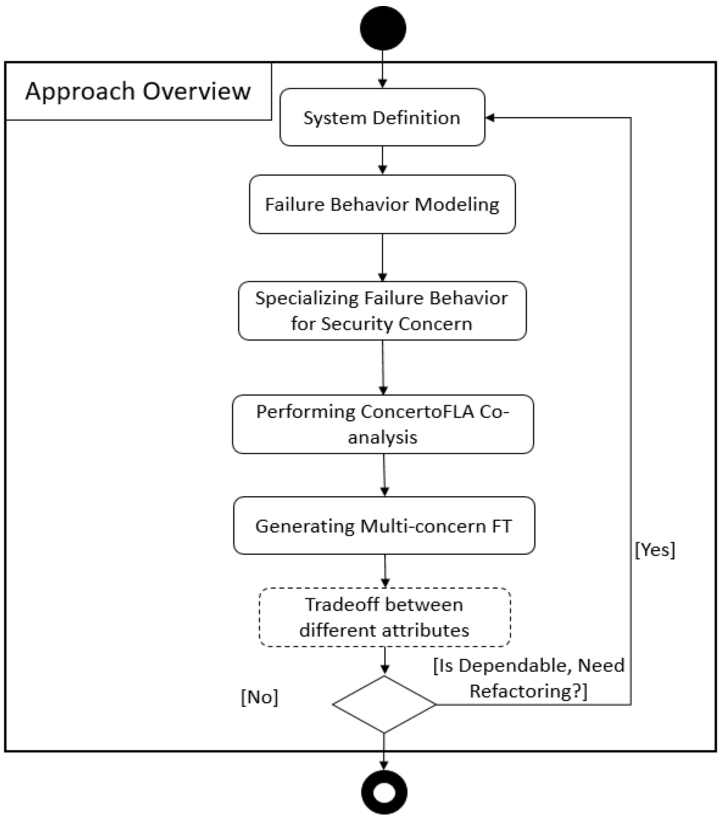

Approach Overview
System Definition
Failure Behavior Modeling
Specializing Failure Behavior for Security Concern
Performing ConcertoFLA Co-analysis
Generating Multi-concern FT
Tradeoff between different attributes
[Yes]
[Is Dependable, Need Refactoring?]
[No]

Figure 52. System Dependability Co-Analysis via ConcertoFLA

In the following subsubsections, the above-mentioned steps are detailed.

3.5.3.2. System Definition

In this first step, expected to be performed by an engineer with architecture engineering expertise, the system is modelled as a composite component. A composite component consists of different composing components and their relationship. Each of the composing components is defined in isolation (independent of composite component for reusability) and has input/output ports for interaction with the environment. To demonstrate the methodology, a simple system is used. This simple system is represented as a composite component named “CompositeDemoSystem”. The system represents a hypothetical controller and is composed of two components i.e., “SensorComponent” and “ControllerComponent”. The former acquires the raw data of a physical phenomenon of the environment on its input port, transforms it into sensor measurement and provides the result on the output port. Whereas the latter takes this measurement on its input port, performs computation and provides the processed data on its output port. This interaction and the above-mentioned functionality of the components is realized through interfaces and component implementations.

To model this simple system, the following steps should be followed:

1. Create a UML Class Diagram under the package “modelComponentView” in Model Explorer view. The diagram allows to define components and other entities e.g., component implementation, interfaces and relationships etc., using the Palette on the right. All components including the composite component are defined in this class diagram. Figure 53 shows all three components i.e., CompositeDemoSystem, SensorComponet and ControllerComponent along with their interfaces and component implementations.

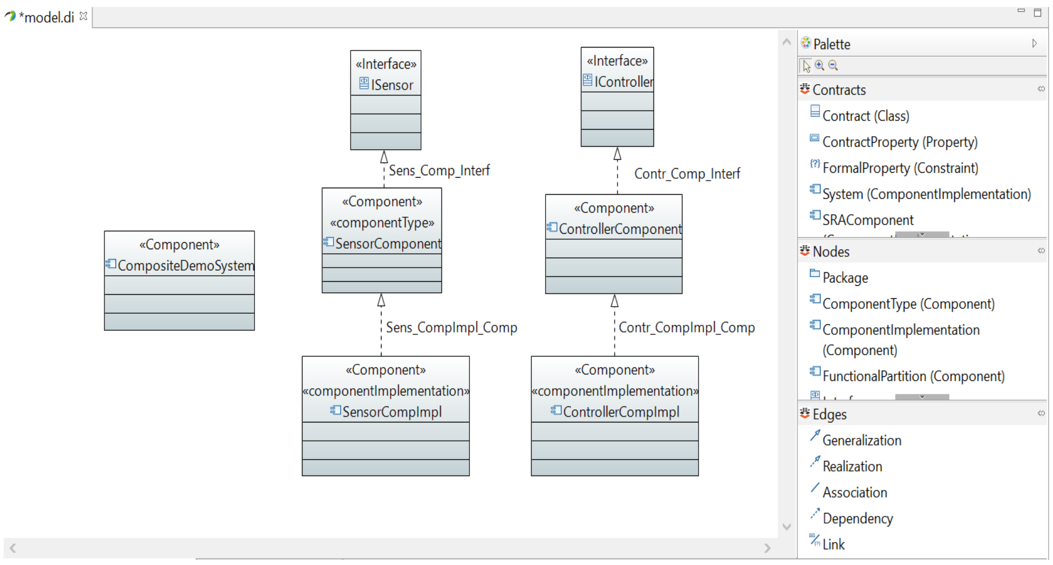

*model.di
«Interface» ISensor
«Interface» IController
Sens_Comp_Interf
Contr_Comp_Interf
«Component» CompositeDemoSystem
«Component» «componentType» SensorComponent
«Component» ControllerComponent
Sens_CompImpl_Comp
Contr_CompImpl_Comp
«Component» «componentImplementation» SensorCompImpl
«Component» «componentImplementation» ControllerCompImpl
Palette
Contracts
Contract (Class)
ContractProperty (Property)
FormalProperty (Constraint)
System (ComponentImplementation)
SRAComponent
Nodes
Package
ComponentType (Component)
ComponentImplementation (Component)
FunctionalPartition (Component)
Edges
Generalization
Realization
Association
Dependency
Link

Figure 53. Component, Interfaces and other entities definition

2. Create a UML Composite Structure Diagram for each of the defined components. This diagram allows to define the input/output ports for the components by enabling “CHESS FunctView” in the Palette. Additionally, other entities e.g., connectors and property etc. could be defined using same view. Figure 54 shows the ControllerComponent being decorated with the input and output ports i.e., sensorData and processedData respectively. Similarly, the SensorComponent is also decorated with its input and output ports.

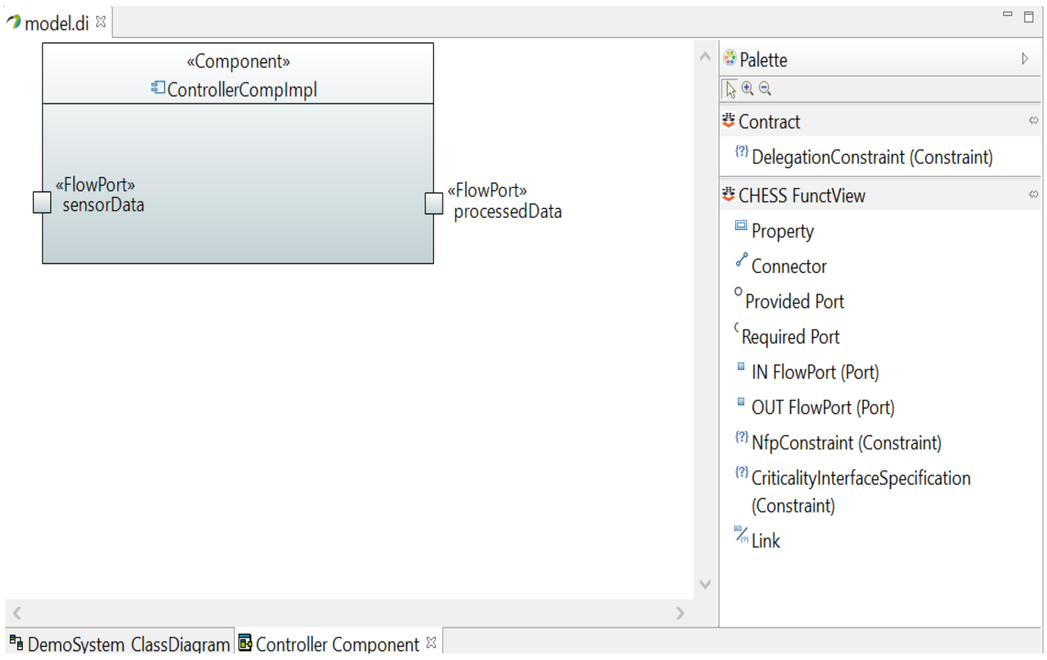


Figure 54. Assigning input/output ports to a component

3. The composite component is also decorated with input/output ports using the editor provided by UML Composite Structure Diagram of the composite component (see Figure 55). Additionally, the composing components and their ports are dragged from the Model Explorer view and dropped in the composite component. The components and ports are then connected using the connector entity from the Palette to realize the system definition. The CompositeDemoSystem component

shown inFigure 55, has one input and output port i.e., rawData and processedData and is composed of “sensorcomponent” and “controllercomponent” which are defined in previous steps.

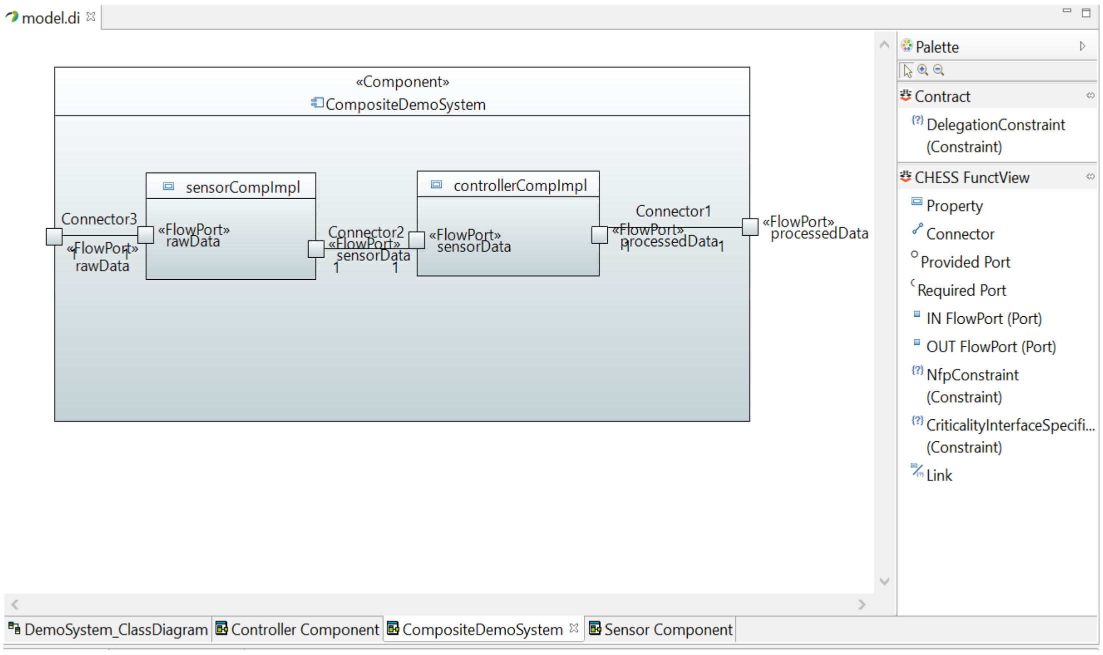
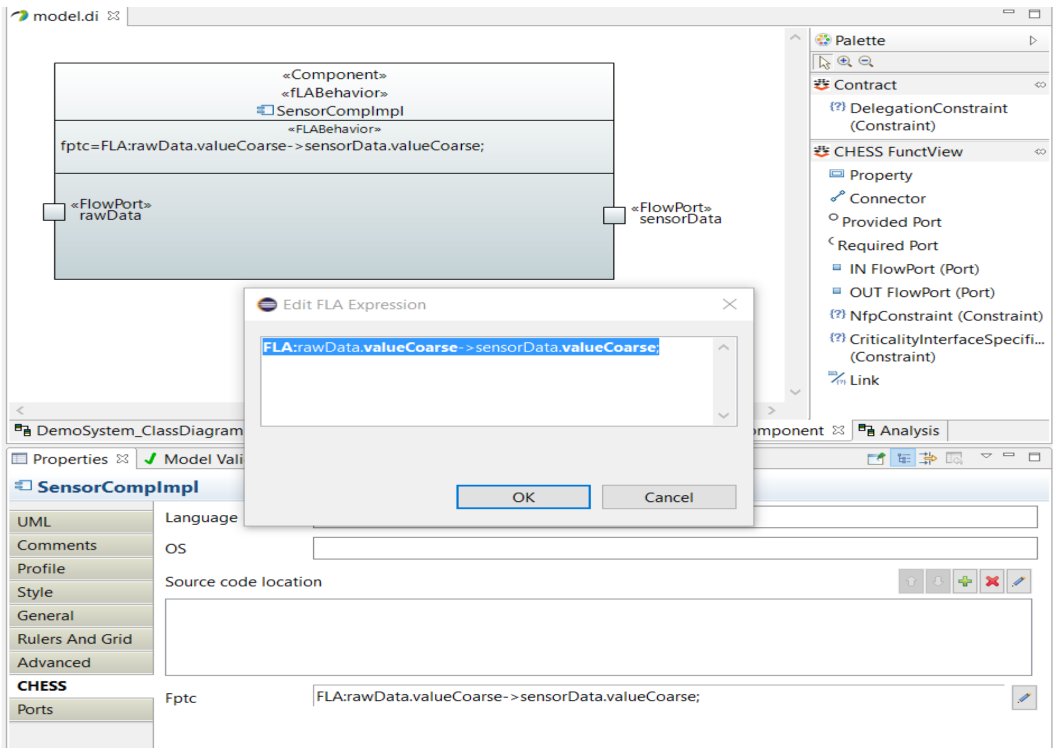


Figure 55. Composite Component

3.5.3.3. Failure Behaviour Modelling

Once, the system is modelled and its components are defined, the safety engineer is expected to model the input-output failure behaviour for each individual component. To perform this modelling, the following step should be followed:

1. Apply “FLABehavior” stereotype to each component and define the failure behaviour using FPTC (Failure Propagation Transformation Calculus) rules (see Figure 56). More specifically, Figure 56 shows one FPTC rule of “SensorComponent”. This rule indicates that if “valueCoarse” (type of fault) is received on the input port called “rawData”, the component propagates valueCoarse to the output port called “SensorData”.



fptc=FLA:rawData.valueCoarse->sensorData.valueCoarse;

FLA:rawData.valueCoarse->sensorData.valueCoarse;

Fptc: FLA:rawData.valueCoarse->sensorData.valueCoarse;

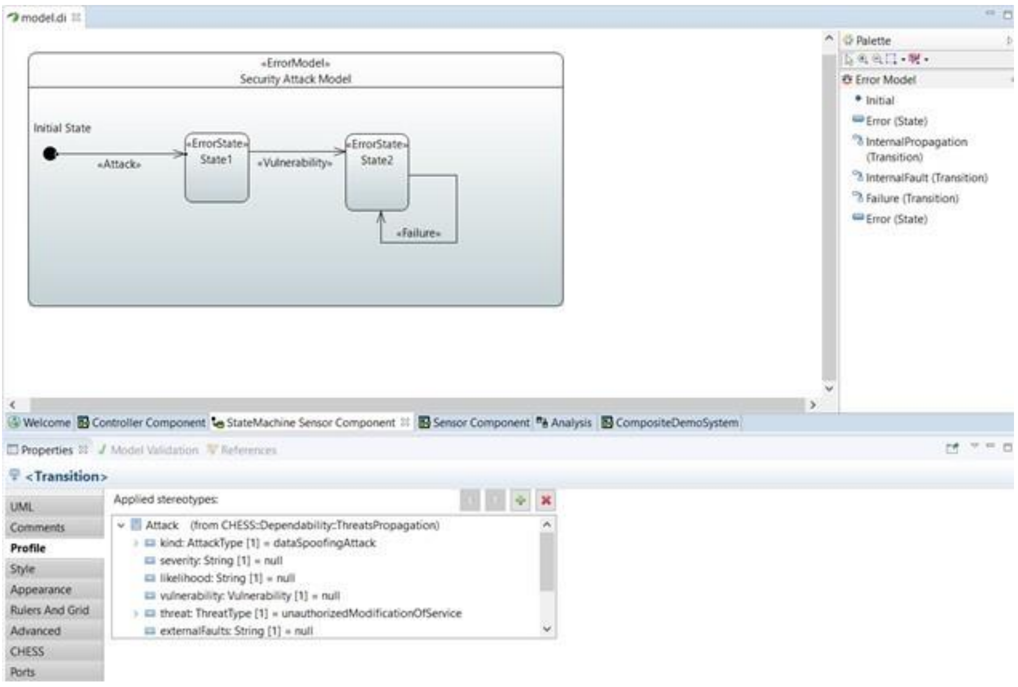
Figure 56. Decorating the components with their failure behaviour

3.5.3.4. Specialising Failure Behaviour for Security Concern

After defining the failure behaviour of each component, a specialisation of this failure behaviour is performed by the security engineer, to address a particular concern e.g., security. This specialisation is achieved via the following steps:

1. Create a StateMachine Diagram under the component, for which the failure behaviour shall be specialized for security concern. Figure 57 shows the state machine diagram created for specializing the “SensorComponent”.
2. Apply “ErrorModel” stereotype to the state machine and define the states and erroneous transitions, using the ErrorModel palette. Figure 57 shows a security attack model for the “SensorComponent”. An attack causes a transition from initial state to the State1. The kind of attack, which depicts its nature, is specified under the “Attack” stereotype along with the threat it would enable if successful. For example, in Figure 57 the “dataSpoofingAttack” is shown, which causes the “unauthorizedModificationofService” threat. The attack exploits a vulnerability to cause a transition to the State2. Similar to the attack, the kind of vulnerability is specified under the “Vulnerability” stereotype. This erroneous transition causes a failure and have a failure mode. This failure mode is specified under the “Failure” stereotype and can be seen in Figure 58. The failure mode is specified as a combination of the port and type of failure i.e., “portname.typeoffailure”, and is similar to the failure behaviour specification shown in Figure 56. The specialization of failure behaviour corresponds to the initial input/output failure behaviour and is meant to enrich its implementation. In this particular case, the failure behaviour of “SensorComponent” is specified as the “valueCoarse” type of failure on “sensorData” port in the presence of “valueCoarse” failure on the “rawData” port. In the context of security, the “dataSpoofingAttack” on the “rawData” port enables the “unauthorizedModificationofService” threat on “sensorData” port due to a “missingDataIntegritySchemes” vulnerability. Each failure of a component can be specialized with corresponding security attack, vulnerability and threat or vice-versa.

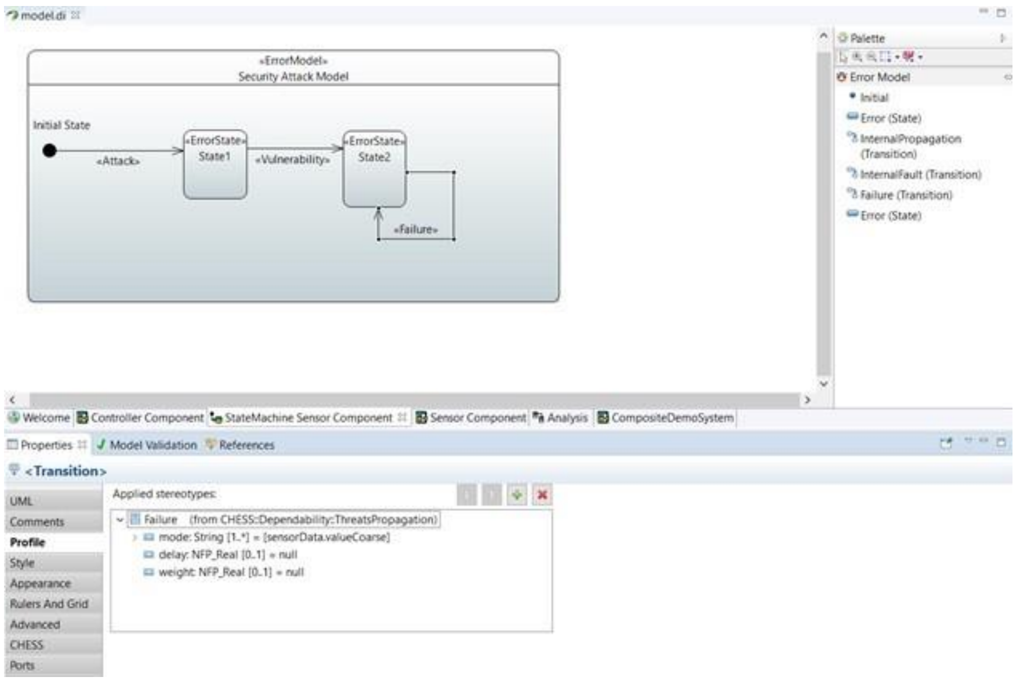
3. Next, “ErrorModelBehavior” stereotype is applied to the component of interest. Figure 59 shows the “SensorComponent” and the “ErrorModelBehavior” stereotype. The “Security Attack Model”, defined in previous step, specializes the failure behaviour of this component.



Applied stereotypes:

```
Attack    (from CHESS::Dependability::ThreatsPropagation)
    kind: AttackType [1] = dataSpoofingAttack
    severity: String [1] = null
    likelihood: String [1] = null
    vulnerability: Vulnerability [1] = null
    threat: ThreatType [1] = unauthorizedModificationOfService
    externalFaults: String [1] = null
```

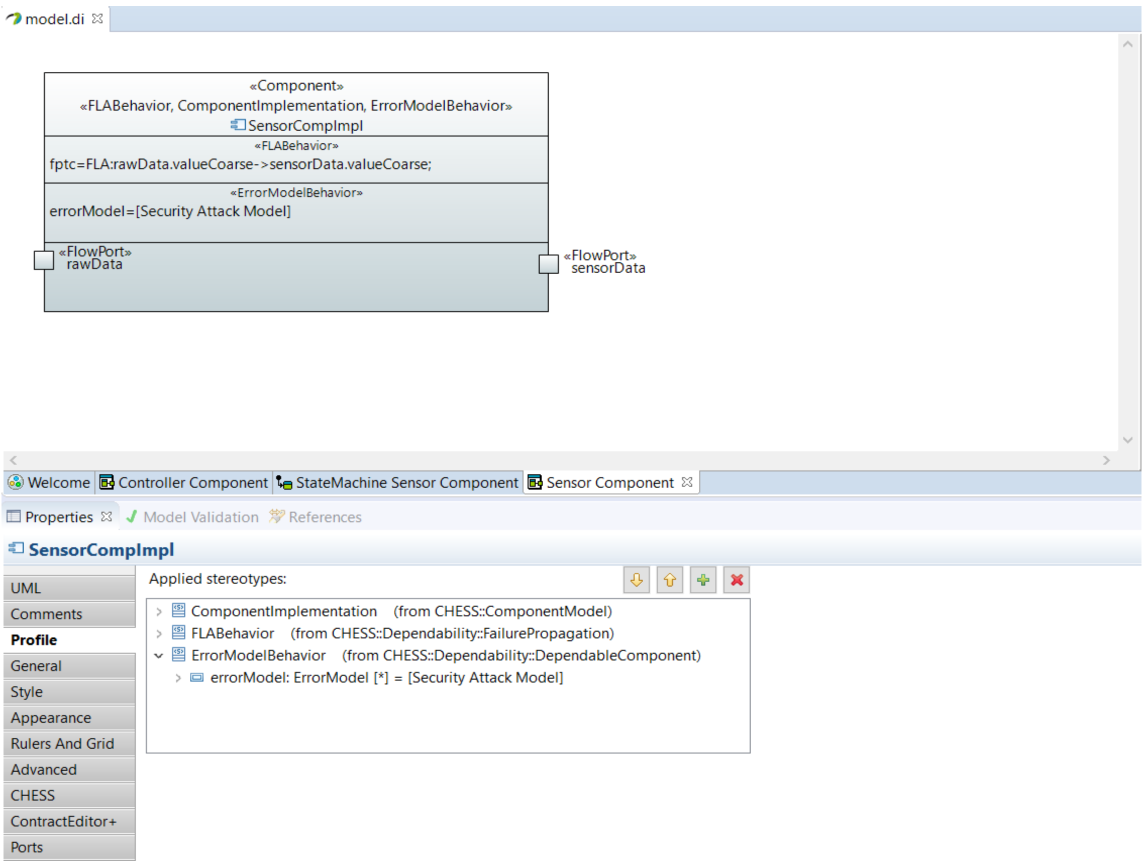
Figure 57. State Machine Diagram illustrating the ErrorModel Stereotyped Security Attack Model



Applied stereotypes:

```
Failure    (from CHESS::Dependability::ThreatsPropagation)
    mode: String [1..*] = [sensorData.valueCoarse]
    delay: NFP_Real [0..1] = null
    weight: NFP_Real [0..1] = null
```

Figure 58. Security Attack Model showing Failure Stereotype State Transition


model.di

«Component»
«FLABehavior, ComponentImplementation, ErrorModelBehavior»
SensorCompImpl
«FLABehavior»
fptc=FLA:rawData.valueCoarse->sensorData.valueCoarse;
«ErrorModelBehavior»
errorModel=[Security Attack Model]
«FlowPort» rawData
«FlowPort» sensorData

Applied stereotypes:
ComponentImplementation (from CHESS::ComponentModel)
FLABehavior (from CHESS::Dependability::FailurePropagation)
ErrorModelBehavior (from CHESS::Dependability::DependableComponent)
errorModel: ErrorModel [*] = [Security Attack Model]

Figure 59. Sensor Component with ErrorModelBehavior Stereotype

3.5.3.5. Performing ConcertoFLA Co-analysis

To perform the ConcertoFLA co-analysis, following steps should be followed:

1. After defining the failure behaviour of the components and specializing it for security concern, the input port of the composite component is annotated with “FPTCSpecification”. The comment is used to specify the type of faults injected to the system. Change the view to “Extrafunctionalview” to enable FPTC drawer in the Palette. Attach FPTCSpecification comment to the input port of composite component. The comment is used to specify the type of faults injected to system. Figure 60 illustrates the type of faults that can be injected in system. The “CompositeDemoSystem” is injected with “valueCoarse” type of fault on its input port (rawData).
2. Create component with “FailurePropagationAnalysis” stereotype in a UML Class Diagram under the package “modelDependabilityAnalysisView” in Model Explorer view. Build the instances of composite component and assign to this newly created component as a resource platform (see Figure 61).
3. Execute the ConcertoFLA co-analysis via “Concerto-FLA co-analysis” menu entry, to generate the failure propagation paths. These paths are stored in a file using Failure Logic Analysis Meta Model (FLAMM) representation. In addition to this, the failures are backpropagated and annotated to the output port of the actual system. Figure 62 shows “CompositeDemoSystem” component with a “valueCoarse” type of failure backpropagated and annotated at its output port (processedData).

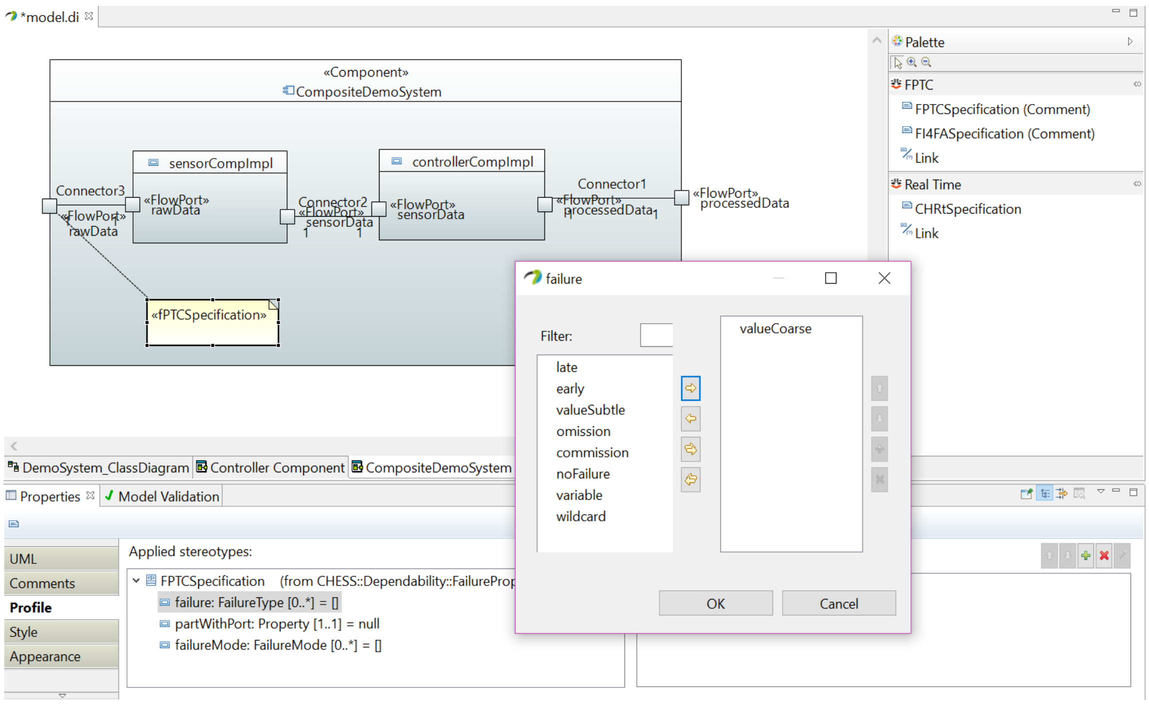


Figure 60. Specifying the injected faults at the input ports of composite component

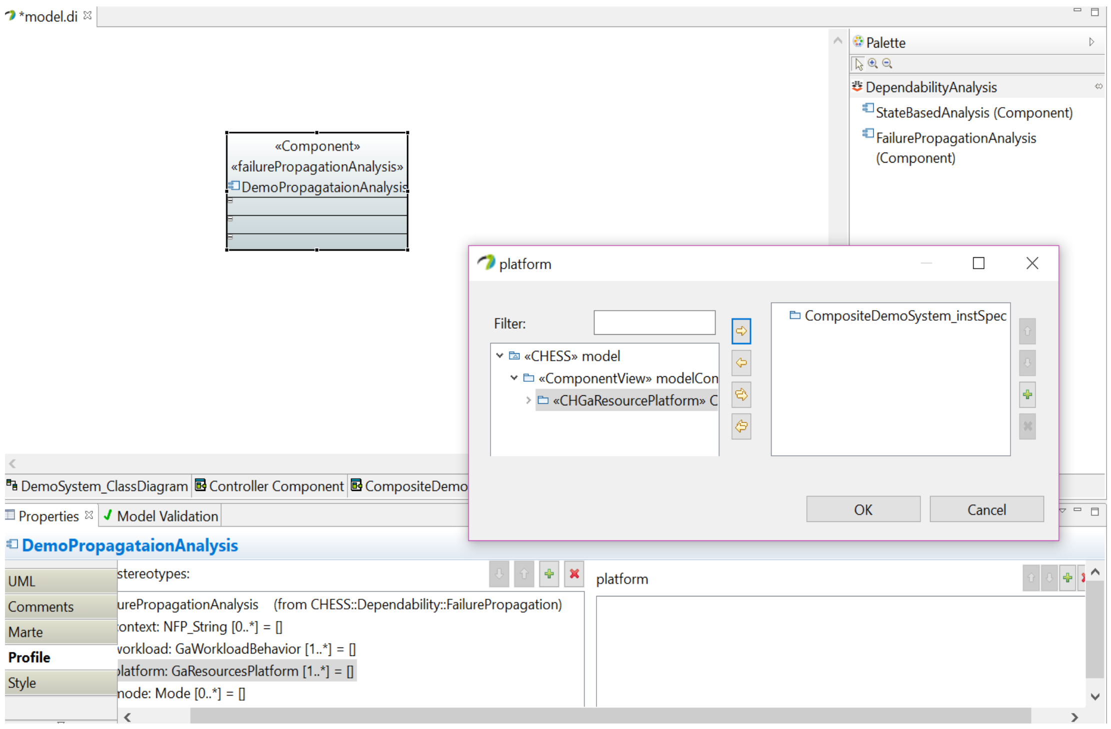


Figure 61. Creating FailurePropagationAnalysis component and assigning resource platform

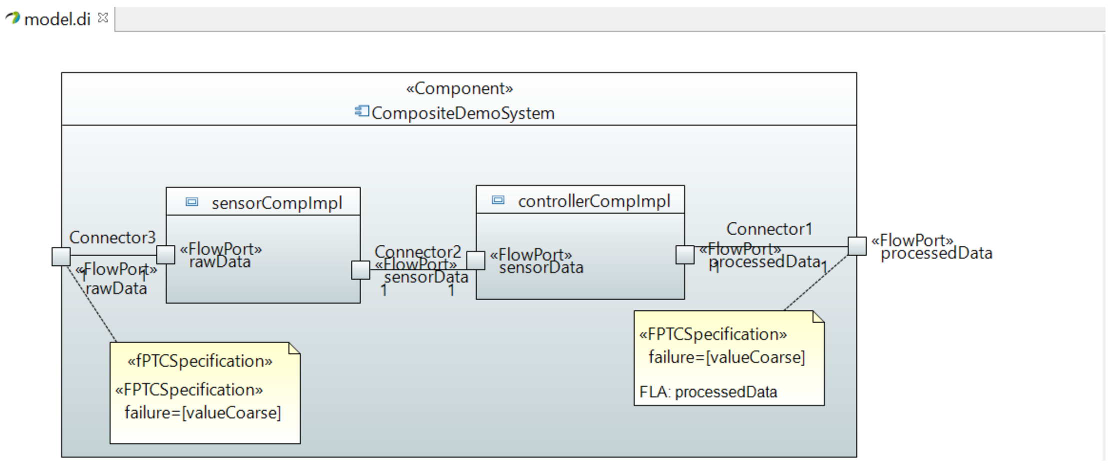


Figure 62. Back-propagated failure on the output port of composite system

3.5.3.6. Generating Multi-Concern Fault Tree

A multi-concern fault tree is generated from the results (failure propagation paths) of Concerto-FLA analysis using following steps:

1. Navigate to the "Generate FT via Concerto-FLA" menu as shown in Figure 63.

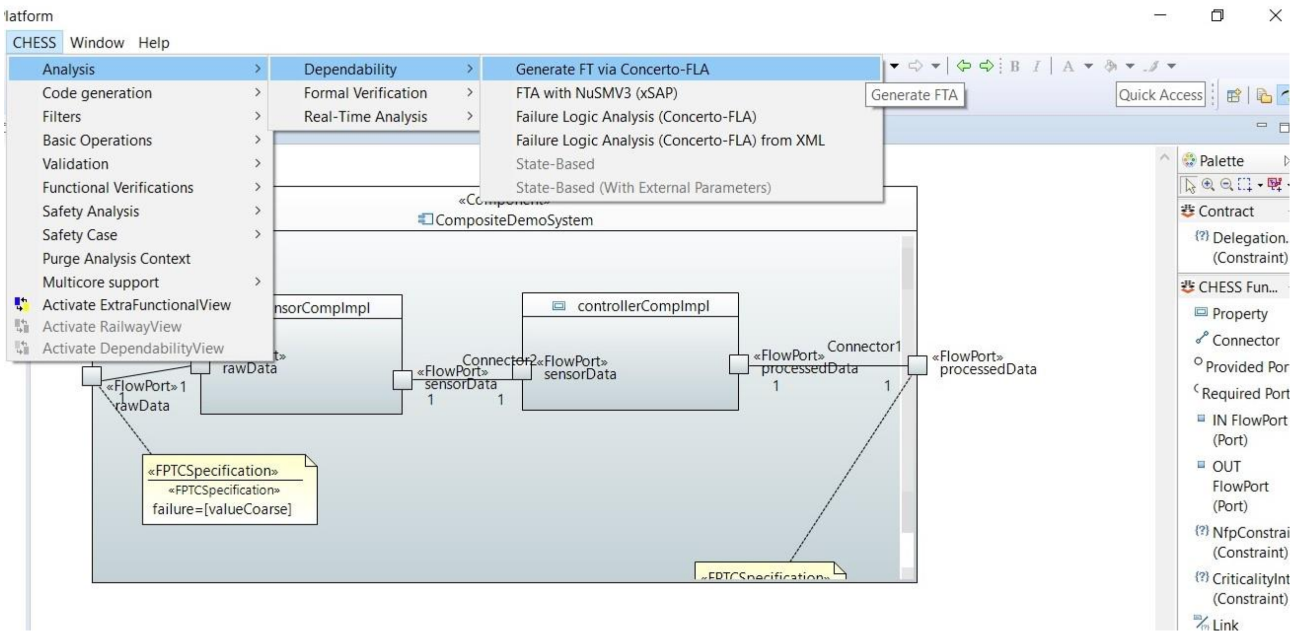


Figure 63. Generate FT via Concerto-FLA menu

2. Upon the selection of this menu a file chooser shall appear. Select the failure propagation paths file generated in previous steps. A corresponding fault tree is generated as shown in Figure 64. Since, the fault tree is generated from the results of failure logic analysis, which supports qualitative analysis, the generated fault tree is also of qualitative nature and does not have probabilities.

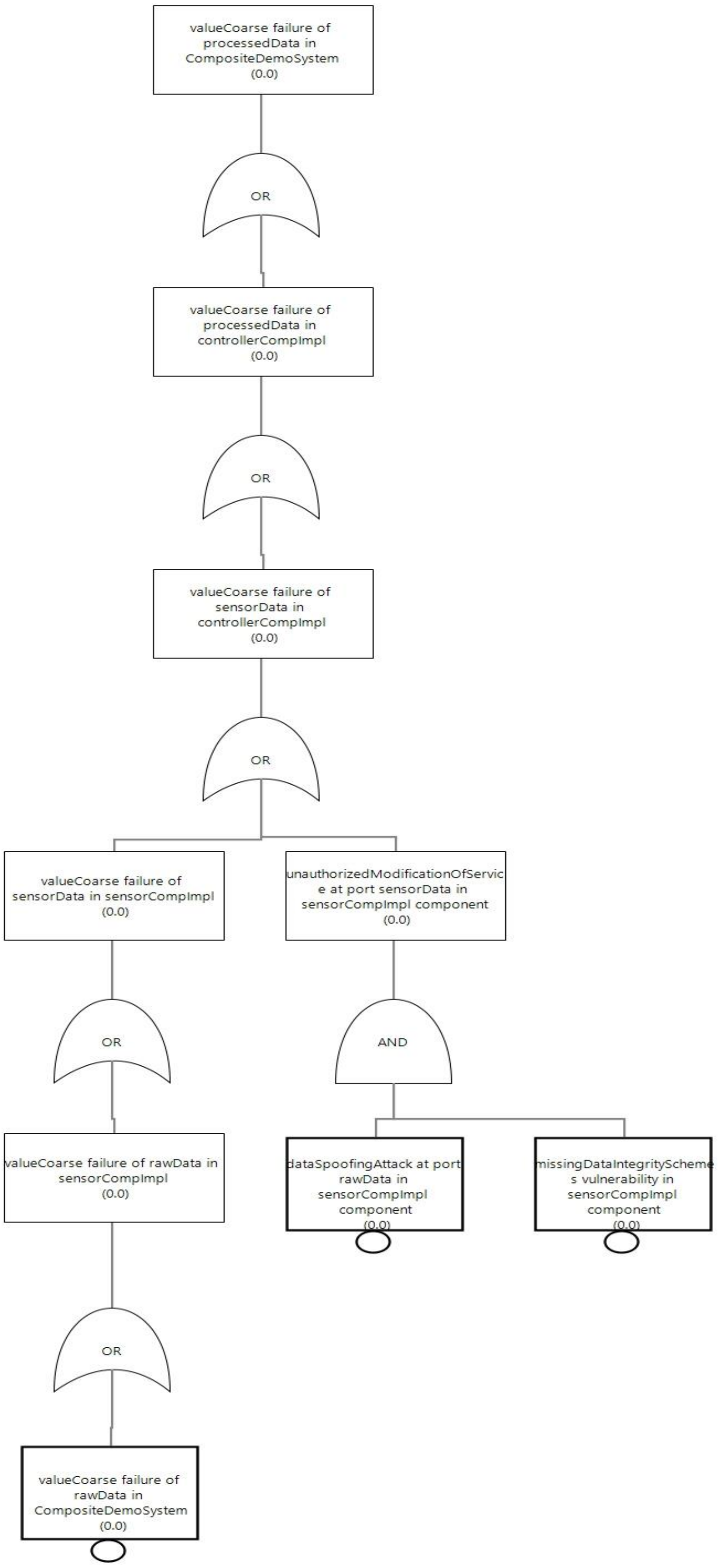

valueCoarse failure of processedData in CompositeDemoSystem (0.0)
OR
valueCoarse failure of processedData in controllerCompImpl (0.0)
OR
valueCoarse failure of sensorData in controllerCompImpl (0.0)
OR
valueCoarse failure of sensorData in sensorCompImpl (0.0)
unauthorizedModificationOfService at port sensorData in sensorCompImpl component (0.0)
OR
AND
valueCoarse failure of rawData in sensorCompImpl (0.0)
dataSpoofingAttack at port rawData in sensorCompImpl component (0.0)
missingDataIntegritySchemes vulnerability in sensorCompImpl component (0.0)
OR
valueCoarse failure of rawData in CompositeDemoSystem (0.0)

Figure 64. Automatically generated multi-concern fault tree

3.5.3.7. Trading Off

Once the multi-concern fault tree is generated, trade-offs can be evaluated by the engineer (architect) together with the safety and security engineer and if needed the system has to be re-designed.

3.6. Privacy Analysis

Dependability of a system can also be considered with regard to how it handles personal data and addresses privacy issues. In this section, we explore how privacy concerns can be tackled by following a methodology and will give two examples. We will first make an overall tour of the relevant privacy concerns. Then we will present the essence of an assurance case-based methodology focusing on privacy concerns. Finally, we will illustrate how existing tools could be used and the requirements in terms of interfacing with the rest of the AMASS framework.

3.6.1. Relevant Privacy Concerns

With the increasing amount of personal data processed at large-scale and the rising of artificial intelligence approaches, privacy concerns driven by public's growing expectations are recognised as gaining importance in the industry. However, privacy is intrinsically vernacular and can designate many different concepts. We will exemplify this by taking a tour of different sources such as regulations targeting privacy. We will then select a couple of privacy concerns from these sources to detail them, show how they are linked, and put the assurance case-based methodology in context.

3.6.1.1. Privacy Regulations

Privacy is differently addressed in the world and obligations incumbent to entities processing personal data might be more or less stringent. For example, in US, federal data protection laws are traditionally sectorial (*Health Insurance Portability and Accountability Act* (HIPAA) for the health sector and *Gramm-Leach-Bliley Act* (GLB) for the finance sector for instance) while there are also state-level regulations (*California Civil Code* §1798.82 for breach notifications for instance) in parallel to many institutional (such as the *Fair Information Practice Principles* (FIPP) recommended by the Federal Trade Commission (FTC)) and private best practices guidelines (such as the cross-industry *Self-Regulatory Principles for Online Behavioral Advertising*) [76]. On the other hand, the approach chosen in most of European countries has been early national legislations later aligned in 1995 through a directive (95/46/EC) in the European Union, which needed transcription into national laws by member states (the Data Act in Sweden in 1973 for instance), sometimes even in their constitution (Portugal in 1976 and Spain and Austria in 1978 for instance).

Recently, a new regulation, the *General Data Protection Regulation* (GDPR, 2016/679) [77], has been voted and entered into force to be applicable today without any need for transcription into national legal laws. This new regulation aims at finding a balance between favouring the free transfer of personal data and providing the adequate protections needed for such transfers and associated processings. It can be noted that the impact of the GDPR is extremely large as it rules, in its Article 3, that not only processings happening in EU are subject to the regulation, but also those processings which, though taking place outside from the EU, concern EU residents. Among other points, the GDPR states the conditions for data protection impact assessments (DPIA) and processings carrying high risks for the data subjects. These two notions will be central to define our methodology. In the following, we will stay close to the GDPR regulatory framework as it is expensive and consistently covers a large economic area.

3.6.1.2. Data Protection Impact Assessment for High Risk Data Processing

In case processing a piece of personal data "is likely to" make the data subject carry a high risk, the data controller (under which the responsibilities and liabilities related to this processing are attached), Article 35 of the GDPR rules that a DPIA should be carried out. In fact, the GDPR let some space here for member states and data controllers: supervisory authorities have been given the possibility to give their positions on the conditions for processings to satisfy this criterion (as did the Belgian authority for instance [78]) and data controllers are free to choose their methodology to perform the DPIA as long as it satisfies established criteria as explained in an opinion from the WP29 [79]. The main goal of all these methods is the same: to help choosing relevant measures to address the risks identified and to demonstrate this has been done for compliance purposes.

3.6.1.3. Organisational and Technical Measures

When risks to rights and freedoms of data subject have been identified through the combination of sufficiently high severity and likelihood, measures aiming at reducing these risks have to be selected. These measures can be of different sorts and belong mainly to two different sorts: organisational measures, on one hand, which target governance, processes and people, and technical on the other hand, which rely on technical components and/or changes to be added to the filing system. A knowledge base of such measures has been published by the CNIL [80], the French supervisory authority, to help data controllers to better protect personal data when designing their systems. Other sources exist to find such measures such as for instance the NIST SP 800-53 and its Security and Privacy Controls which can act as a guide and support knowledge base to find ways to engineer privacy in a system [81].

3.6.1.4. Accountability through Compliance Demonstration

Beyond mere data protection by design as imposed by Article 25 of the GDPR, the new standard is to be able to also demonstrate compliance (a principle called “accountability”) set out in Article 5 of the GDPR. To comply with this obligation, it is needed to document all the elements which are artefacts of the data protection related aspects of a system processing such data. In particular, the DPIA has to be carefully driven and decisions made based on its outcomes must be well motivated. In this context, an assurance case can help to show how arguments are supported and by which claims and evidences.

3.6.2. Privacy Assurance Case Methodology

3.6.2.1. Privacy-Related Requirements

Regulations tend to be prescriptive corpus which can be considered as requirements which should be satisfied by economical actors. Concerning data protection, this still constitutes a challenge as the GDPR, for instance, is a long and dense regulation which could be decomposed in many requirements, stratified at different levels to organise them all. We take here as an example a series of requirements taken from the GDPR. Generating such requirements in a structured way constitutes a field already addressed by other works [82][83].

We take a simple example that we will develop in the following to show how an assurance case can be used for data protection purposes. One of the keystones of data protection standards and regulations is the concept of purpose limitation. This is an indispensable requirement which states in Article 5§1(b) of the GDPR:

“[...] personal data shall be […] collected for specified […] purposes and not further processed in a manner that is incompatible with those purposes […]”

Once such a requirement has been identified, it has to be met by the implementation and this should be reflected in unit/integration test/verification cases results.

3.6.2.2. Requirements Traceability

In order to demonstrate compliance to regulations through satisfaction of requirements, the traceability of these latter has to be ensured between multi-layer artefacts. The traceability can for instance link a high-level requirement to several low-level requirements, or requirements to verification cases. Coming from the development of critical systems area, such as avionics or automotive, such approaches are also relevant for systems dealing with highly sensitive personal data. There is also a need to maintain the traceability between these requirements and the solution chosen to satisfy them. Contracts can be used to this aim to formalize privacy requirements and ensure the architectural level consistency.

3.6.2.3. Privacy Assurance Case and Evidences

The demonstration of which solutions can be used for evidences to satisfy a requirement can be made through an assurance case. For example, Figure 65 shows an assurance case in the context of the GDPR. The root targets compliance with Article 5§1 while the requirement expressed above is the sub-goal G5.1.b.

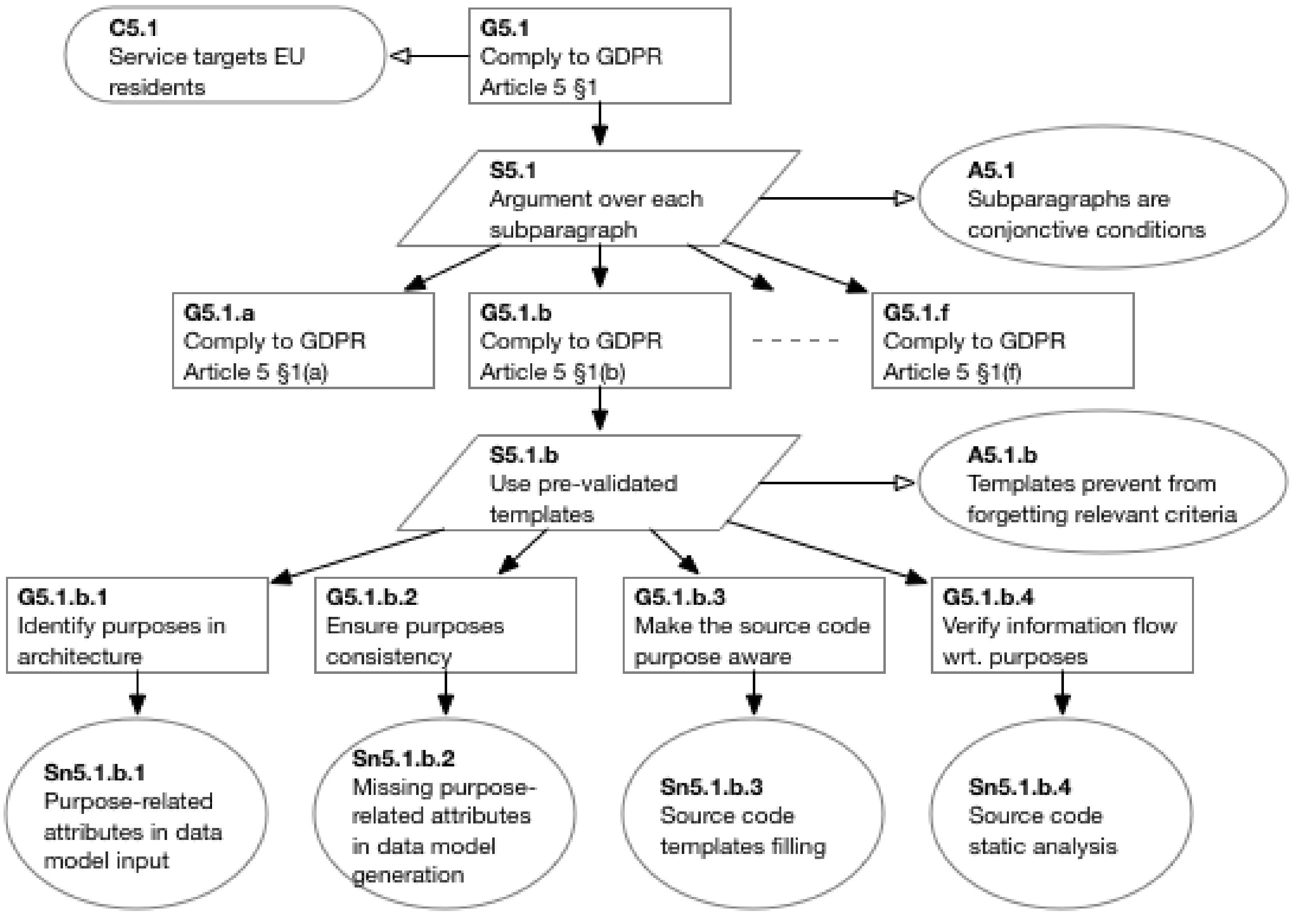

C5.1
Service targets EU residents
G5.1
Comply to GDPR Article 5 §1
S5.1
Argument over each subparagraph
A5.1
Subparagraphs are conjonctive conditions
G5.1.a
Comply to GDPR Article 5 §1(a)
G5.1.b
Comply to GDPR Article 5 §1(b)
G5.1.f
Comply to GDPR Article 5 §1(f)
S5.1.b
Use pre-validated templates
A5.1.b
Templates prevent from forgetting relevant criteria
G5.1.b.1
Identify purposes in architecture
G5.1.b.2
Ensure purposes consistency
G5.1.b.3
Make the source code purpose aware
G5.1.b.4
Verify information flow wrt. purposes
Sn5.1.b.1
Purpose-related attributes in data model input
Sn5.1.b.2
Missing purpose-related attributes in data model generation
Sn5.1.b.3
Source code templates filling
Sn5.1.b.4
Source code static analysis

Figure 65. Example of data protection assurance case

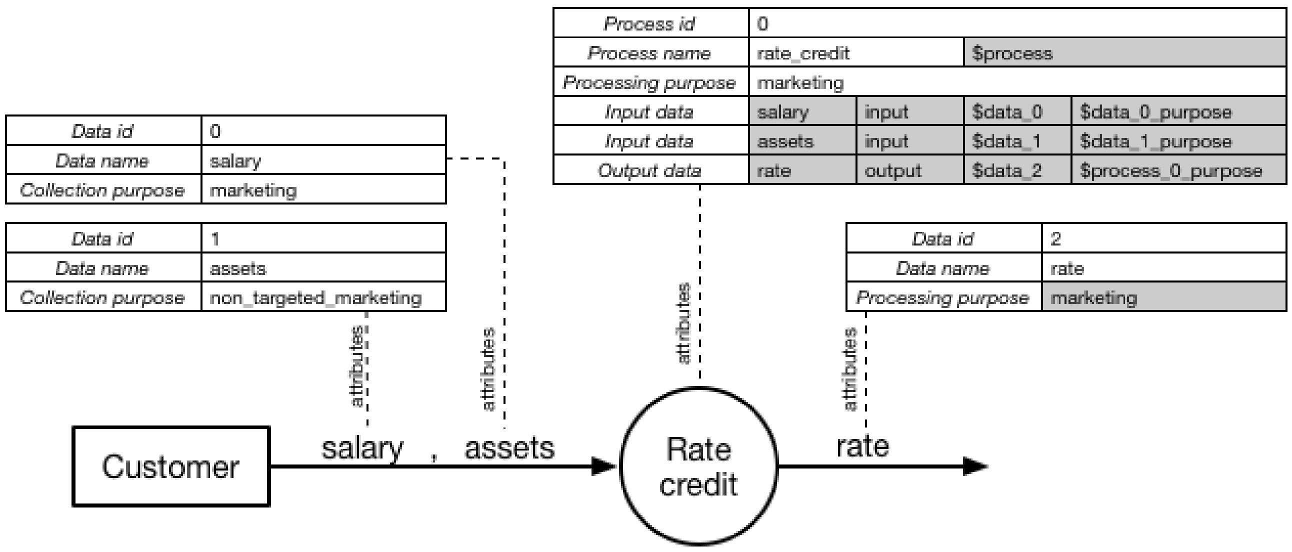
It can be seen that goal G5.1.b is split into four other sub goals which are linked to solutions. These solutions cover several layers of the system thus benefitting from the multi-concern approach of AMASS since it will build on some elements already defined for other concerns.

3.6.3. Verification of Privacy-Related Requirements

In the context of the AMASS project, the Frama-C software analysis platform can be used to address privacy-related concerns. Our method is inspired from the way to model privacy purposes suggested by [84] applied to Secure Flow [85], a plug-in from the Frama-C software analysis platform [86]. This plug-in allows to reason about information flow in a C program and can be leveraged for privacy purposes.

3.6.3.1. Architecture-Level Privacy-Related Requirements

The requirements at the architectural level are expressed in formal contracts. A data flow-oriented view of the architecture can be used and augmented with specific privacy-related attributes for some of its elements. An example of a concrete case is shown in Figure 66.



Process id	0			
Process name	rate_credit		$process	
Processing purpose	marketing			
Input data	salary	input	$data_0	$data_0_purpose
Input data	assets	input	$data_1	$data_1_purpose
Output data	rate	output	$data_2	$process_0_purpose

Data id	0
Data name	salary
Collection purpose	marketing

Data id	1
Data name	assets
Collection purpose	non_targeted_marketing

Data id	2
Data name	rate
Processing purpose	marketing

Figure 66. Architectural view of the data flow diagram and its attributes (white values correspond to Sn5.1.b.1 and grey values correspond to Sn5.1.b.2 from Figure 65)

This small architecture shows an external entity (Customer) which send two pieces of data (*salary* and *assets*) to a process (*Rate credit*) before this latter sends an output value (*rate*) to another part of the system (not shown in this example). This set of basic elements constitutes a collection of personal data and is thus subject to the GDPR and in particular its purpose limitation principle which corresponds to the requirement described above.

In order to deal with privacy concepts, the classical view has been augmented with attributes attached to each its constituting elements. They are manually input by the designer in the model design software and define to identifiers (*Data id* and *Process id*), names (for *Data* and for *Process*), and purposes (for *Collection* and for *Processing*). At this stage, the purposes need to be ordered by their permissivity. For example, it can be considered that processing data with a *marketing* purpose is more permissive than with a restriction to a *non_targeted_marketing* only. This relation can be defined as shown in Equation 1. Filling in these attributes values, which have a white background in Figure 66, and defining the ordering of purposes correspond to solution Sn5.1.b.1 from Figure 65.

```
"non_targeted_marketing" ≤ "non_targeted_marketing"
"marketing" ≤ "marketing"
"marketing" ≤ "non_targeted_marketing"
```

Equation 1. Definition of the ≤ purposes ordering

The following step is to reason about the architecture to deduce new attribute values with the information input. For instance, the *Processing purpose* of the data *rate* is inherited from the *Processing purpose* of the process *Rate credit*. Similarly, the *Input* and *Output data* from the *Rate credit* process are generated from its input datas in the architecture and qualified as *input* or *output* before being assigned variable beginning with a *$* sign, acting as placeholders, which will be used to make the link with the source code at a later stage. The result of this generation, which corresponds to solution Sn5.1.b.2 from Figure 65 is denoted with a grey background in the tables in Figure 66.

3.6.3.2. Code-Level Privacy-Related Requirements

Two main steps are considered at code-level before performing the verification: a program corresponding to the architecture has to be written on one hand and the privacy-related requirements have to be specified on the other hand.

3.6.3.2.1. Program Generation

The development of a program which will be verified with regard to the privacy properties expressed above will be based on code generation. The C code which will be generated will mainly handle two kinds of aspects: inputs and outputs. The template used for inputs is shown in Code 1 where the name of this specific data is declared and assigned to name ***input_i***, a specific purpose is declared and assigned to ***input_purpose_i***, this purpose is then added to a key-value dictionary ***purposes*** storing the link between data and purposes, a comment line which will be later (explained and) used for information flow control is added, and an integer value is input from outside through a call to a function called ***read_input_i_value***. The letter ***i*** contained in variable names (***input_i***, ***input_purpose_i***, and ***input_i_value***) is a placeholder for the identifiers ***Data id*** visible on Figure 66 (this remain the case throughout this section without any new mention of it). Please note that there are also strings beginning with ***$*** (such as ***$data_i*** and ***$data_i_purpose***) and which also corresponds to their counterpart from the architectural level from Figure 66 .

```
char input_i[] = "$data_i";
char input_purpose_i[] = "$data_i_purpose";
insert(purposes, input_i, input_purpose_i);
//@
int input_i_value = read_input_int();
```

Code 1. Template for input purpose limitation

The template used for output data is shown in Code 2. It follows the same principles as for the input data. It can be noted it calls another function which consists in processing personal data before returning its result.

```
char output[] = "$data_i";
char output_purpose[] = "$process_i_purpose";
insert(purposes, output, output_purpose);
int output_value = $process(input_i_value, …);
//@
return output_value;
```

Code 2. Template for output purpose limitation

Given the architecture from Figure 66 which contains two inputs and one output data, the consolidated template shown in Code 3 can be generated by instantiating the *i* corresponding to the identifiers of the data elements.

```
int main(void)
{
    char input_0[] = "$data_0";
    char input_0_purpose[] = $data_0_purpose;
    insert(purposes, input_0, input_0_purpose);
    //@
    int input_0_value = read_input_int();

    char input_1[] = "$data_1";
    char input_1_purpose[] = $data_1_purpose;
    insert(purposes, input_1, input_1_purpose);
    //@
    int input_1_value = read_input_int();

    char output[] = "$data_2";
    char output_purpose[] = $process_0_purpose;
    insert(purposes, output, output_purpose);
    int output_value = $process(input_0_value, input_1_value);
    //@
    return output_value;
}
```

Code 3. Consolidated template for main function of *Rate credit* application

This process satisfies solution Sn5.1.b.3 from Figure 65 about template filling to preserve architectural properties at the code-level.

3.6.3.2.2. Privacy Specification

As mentioned above, this consolidated template contains comment locations (lines beginning with *//@*) which are places in which a formal specification will be input. This specification is expressed in a formal language called ACSL [87], which will be used by Frama-C/SecureFlow to verify the C implementation meets the specification. The privacy-related requirements thus need to be translated in ACSL such that SecureFlow can use them. The current version of SecureFlow follows a pattern commonly used by information flow control tools which relies on the annotation of variables with annotations denoting their level of confidentiality. These two levels are *private* and *public*. To declare a variable to be *private*, a comment *//@ private* must be added at the line preceding its declaration in the source code; if not done, the variable is considered as *public*.

In terms of privacy-related requirements, the output data annotation will be *public* and data associated with restrictive purposes will be set as *private*. It will then be verified if some data for which only a restrictive purpose has been allowed are used or not to compute output data (considered to have a non-restrictive purpose by definition). The algorithm corresponding to this is shown in Algorithm 1.

```
for 0 ≤ i < n:
    if (lookup(purposes, input_i) ≤ lookup(purposes, output)):
        input_i_value.status <- public
    else:
        input_i_value.status <- private
output_value.status <- public
```

Algorithm 1. Purpose limitation ACSL specification generation

This algorithm calls a function called *lookup* which applies to a dictionary and is the counterpart of *insert* introduced in input and output templates: it searches for the value corresponding to a key. The assignment of the *public* or *private* annotation is specified as a status of the variable and is done through the sign <-.

Following the example and the purposes indicated in Figure 66, the output *rate* will be *public*, the input *salary* will be *public* (as it has a permissive purpose), and *assets* will be *private* (as it is restricted to *non_targeted_marketing* only). This result is combined with the filling of the template coming from Code 3 where all *$*-variables are substituted by values coming from the architecture (see Figure 66). The result of this is shown in Code 4. The latest comment *//@ assert* is written in this long form in order to trigger the information flow verification.

```
int main(void)
{
    char input_0[] = "salary";
    char input_0_purpose[] = "marketing";
    insert(purposes, input_0, input_0_purpose);
    //@
    int input_0_value = read_input_int();

    char input_1[] = "assets";
    char input_1_purpose[] = "non_targeted_marketing";
    insert(purposes, input_1, input_1_purpose);
    //@ private
    int input_1_value = read_input_int();

    char output[] = "rate";
    char output_purpose[] = "marketing";
    insert(purposes, output, output_purpose);
    int output_value = rate_credit(salary_value, assets_value);
    //@ assert security_status(output_value) == public;
    return output_value;
}
```

Code 4. Purpose-limited main function of *Rate credit* application

The specification of the program made following this method prepares the satisfaction of solution Sn5.1.b.4 from Figure 65. In order for this verification to be performed, it is necessary to add a couple of other annotations and content in the code. First, all functions called need to be, at least, abstractly described for the SecureFlow to be able to deduce information flow properties from the main program. An ACSL specification is thus added which indicates which variables are used to assign a value to the return value of the function. For instance, for the function *read_input_int*, a technical variable **__fc_stdin* (denoting data coming from the standard input) is used to compute *\result* as shown in Code 5.

```
//@ assigns \result \from *__fc_stdin;
char* read_input_int(void);
```

Code 5. Flow-oriented ACSL specification of `read_input_int` abstract function

The same is done for the *insert* function which serves to populate the *purposes* dictionary as shown in Code 6.

```
//@ assigns table \from table, key, value;
void insert(dict table, char key[], char value[]);
```

Code 6. Flow-oriented ACSL specification of `insert` abstract function

For the sake of completeness, the body of the *rate_credit* function must be filled and will be used by SecureFlow to perform its verification. An example of this is shown in Code 7 where the content of *assets* is only used if the *salary* is not enough.

```
int rate_credit(int salary, int assets) {
    int rate = salary * 12 / 100;
    if (rate < 200) { rate += assets / 250; }
    return rate;
}
```

Code 7. Definition of `credit_rate` function

3.6.3.3. Traceability to Architecture Level

The verification performed by SecureFlow in this specific example indicates that the content of a *private* variable flows towards a *public* variable. This is because *assets* has been associated a restrictive purpose with regard to the purpose for which the data is processed by *credit_rate* (which is a permissive purpose associated to its output value *rate*). This verification completes the use of sequential solutions suggested in the privacy assurance case described in Figure 65 and the artefacts from the verification can be used as evidences of contracts satisfaction at the architecture level. This will be used to demonstrate compliance to the GDPR and may also be used for other concerns (for instance for security properties related to confidentiality issues).

4. Cases Studies

In this chapter, some case studies are used to illustrate the execution of the workflows presented in the Chapter 3. In particular, the AMASS CS11 is used to illustrate dependability co-analysis via ConcertoFLA; the AMASS CS3 is used to illustrate the dependability multiconcern assurance approach; and the AMASS CS1 is used to illustrate the dependability co-analysis via Safety Architect. Finally, the normative documents used in the automotive domain are used to illustrate the process-related dependability co-assessment.

4.1. Case Study CS11 - Attitude and Orbit Control System (*)

Attitude and Orbit Control System (AOCS) is a satellite-on board application that provides two functionalities: 1) attitude control, which controls the orientation of a satellite relative to a reference frame (e.g., celestial bodies such as Sun and Earth etc.) and, 2) orbit control, which controls the position of a satellite in an orbit. AOCS collects the attitude data from the attitude sensors and calculates control torques to be applied on satellite using the actuators to achieve desired attitude and position in orbit.

European Cooperation for Space Standardization (ECSS) provides standards for engineering, management and qualification of a space system, more specifically, ECSS-E-ST-40C [50] for software engineering and ECSS-Q-ST-80C [51] for product assurance. ECSS-Q-ST-80C defines different criticality levels for a system based on the severity of consequences of the failure. AOCS is categorized as critical system due to the severe consequences of its failure and is required to fulfil the requirements applicable through ECSS throughout its engineering process.

To this end, this case study is focused only on the attitude control function which will be addressed as Attitude Control System (ACS) hereafter. ACS controls the orientation of satellite by applying the torques through *attitude actuators* (reaction wheel and/or thrusters) in a closed loop over following steps:

- Reading data from the attitude sensors.
- Estimating the current attitude of the satellite relative to the reference frame of interest.
- Calculating the deviation from the targeted attitude.
- Calculating the control torque to minimize and converge to the targeted attitude.
- Generating and sending the commands to the attitude actuators to apply the computed torque on satellite.

ACS has different functioning modes which correspond to different pointing requirements. These pointing requirements are formulated according to the satellite mission and its objectives. For example, ACS in safe mode is required to point its solar panels towards the Sun to power up all its critical parts, thus controlling the attitude relative to the Sun. Similarly, ACS in *mission mode* for a telecommunication mission needs to control the attitude relative to the Earth.

4.1.1. Description of the Use Case Scenario

We have considered a simple use case of ACS i.e., ACS in *Sun Pointing mode* to demonstrate the co-analysis methodology. In this mode, ACS controls the attitude of a satellite relative to the Sun as the requirement is to point to the Sun. ACS requires a sun sensor to acquire the attitude data, which provides the direction of the Sun in the sensor's reference frame. Attitude of the satellite is estimated from this attitude data and is represented as an estimated sun vector, which depicts the satellite orientation in sun reference frame. Estimates for the angular velocity of the satellite are computed from the measured angular rates using a gyroscopic sensor. Additionally, gyroscopic disturbance torque is also calculated from these angular rates to compensate for gyroscopic coupling in dynamics.

In this use case, the generation of actuation commands to apply the computed control torque on the satellite is considered out of the scope.

4.1.2. Demonstration of the Methodology

We start with the system definition and model five components to fulfil the above-mentioned ACS functions, as illustrated in Figure 67.

As done in [34] and [72], the ACS system is defined as a composite component, which contains the following components:

- SignalConditioner, which process and transforms sensor data to satellite reference frame.
- StateEstimator, which estimates the satellite state using the current state measurements and historical data.
- PDController, which calculates the proportional and derivative torque using estimated sun vector and angular velocities.
- SteerController, which calculates the torque using only the estimates of sun vector.
- FeedforwController, which computes the gyroscopic coupling torque.
- TorqueSelector, which selects the appropriate torque from the torques computed by PDController and SteerController, as the control torque based on the current attitude of satellite.

A detailed description of each of the components and the difference between the controllers can be found in [34], which represents an initial step towards a complete case study of AOCS (CS11 in D1.1 [1]). All of the above-mentioned components along with the ACS composite component are defined along with their interfaces, implementations and other entities and is shown in Figure 67.

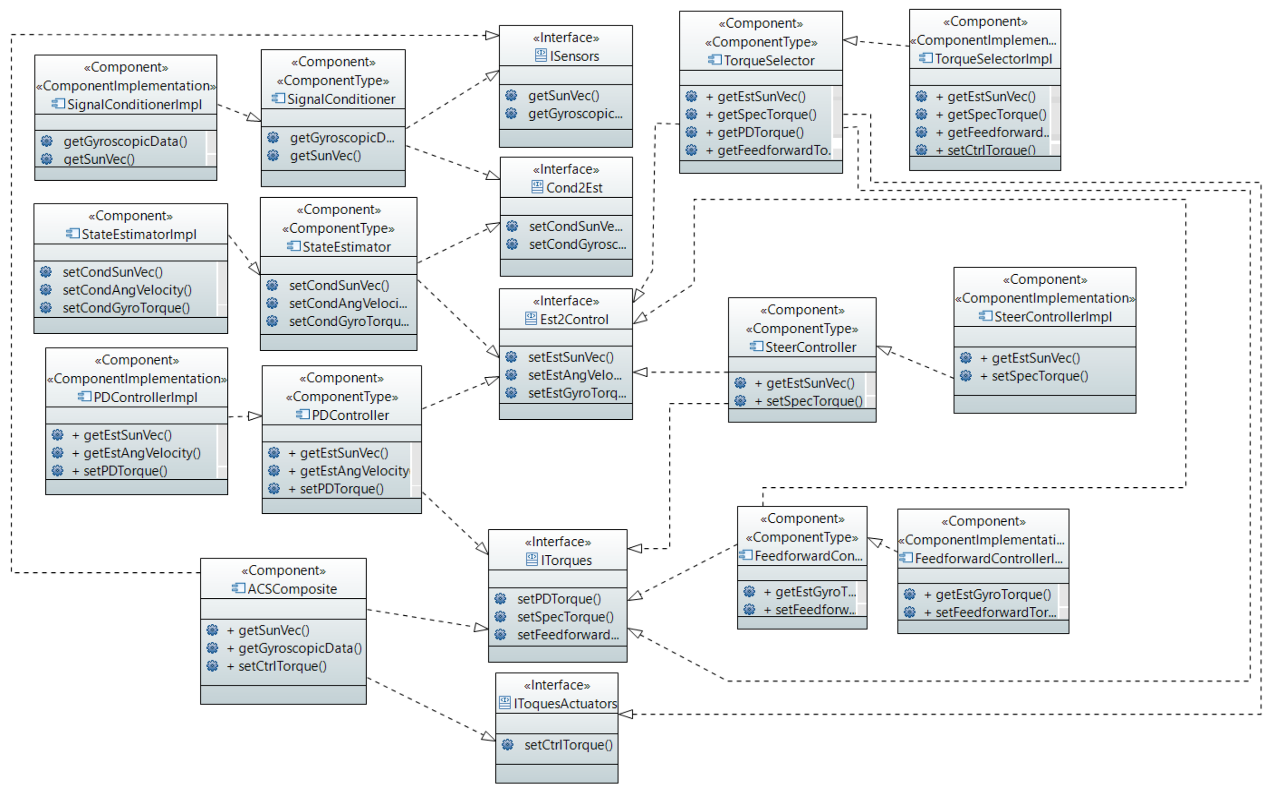


Figure 67. Class diagram showing the components of the ACS system

Next, input and output ports for all these components are defined along with their failure behaviours. And then, all of these components are connected together to compose the ACS composite component as shown in Figure 68. In the next step, the failure behaviour for all components is specified. In this use case, we consider the “valueCoarse” type of failure and specified the failure behaviour only for this case – as motivated in [33]. This failure on the input port of ACSComposite i.e., SunVec corresponds to the scenario, where the Sun Sensor is providing incorrect measurements. To address the security concern, we model a scenario that these incorrect values are due to a data spoofing attack and are not actually originated from the Sensor unit. To demonstrate this, a state machine (Error Model) is created for the “SignalConditioner” component, which shows the “dataSpoofingAttack” along with the “missingDataIntegritySchemes” vulnerability and causing an “unauthorizedModificationOfService” threat. This has been demonstrated in the Figure 69. Once, the failure behaviour and specialization is modelled, the input ports of the composite component are injected with the faults and failure logic analysis is executed. As a result of the execution, the calculated failure behaviour is back-propagated to the initial input model and stored in the FLAMM based file.

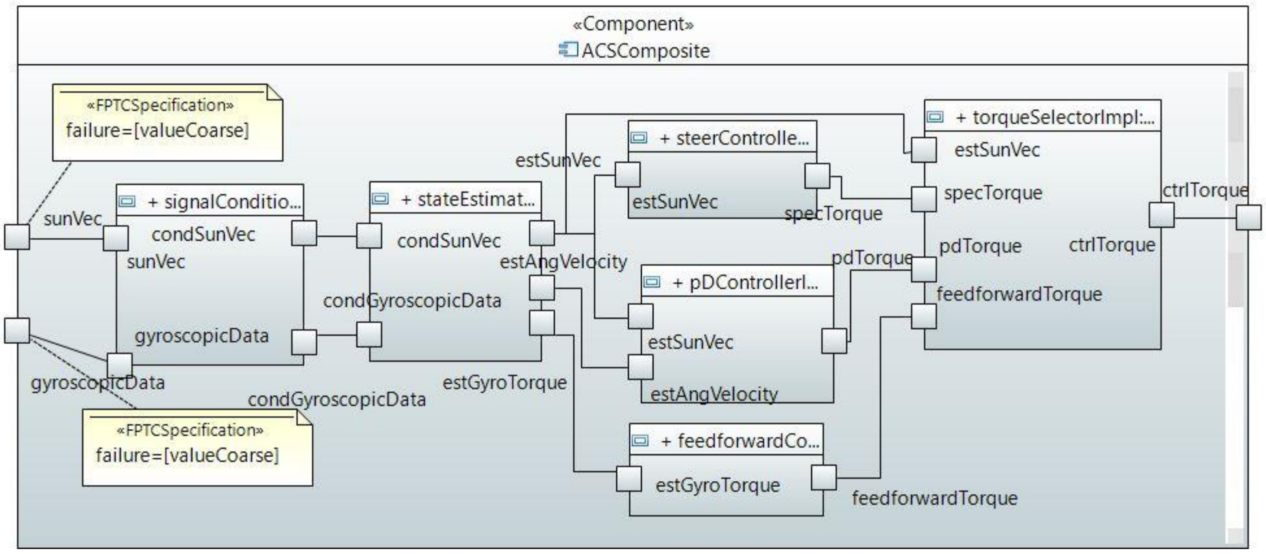


Figure 68. ACS Composite component

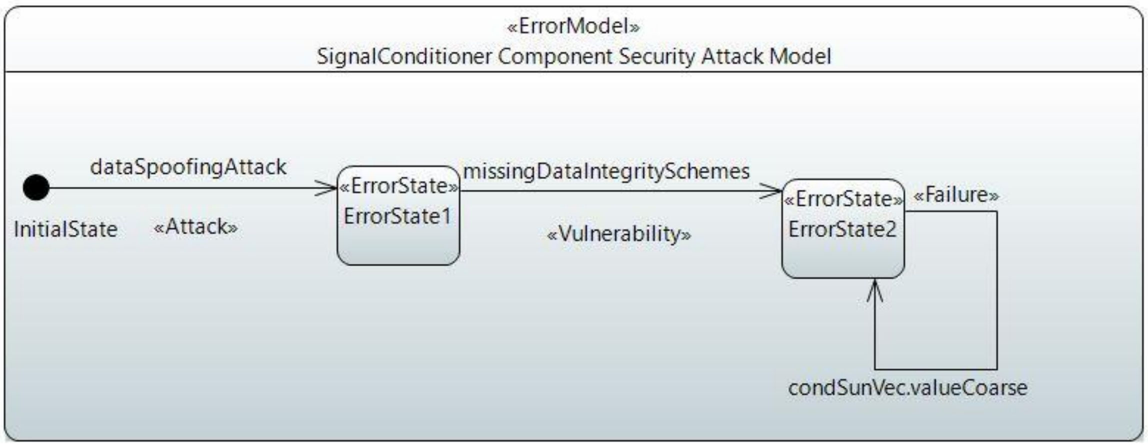
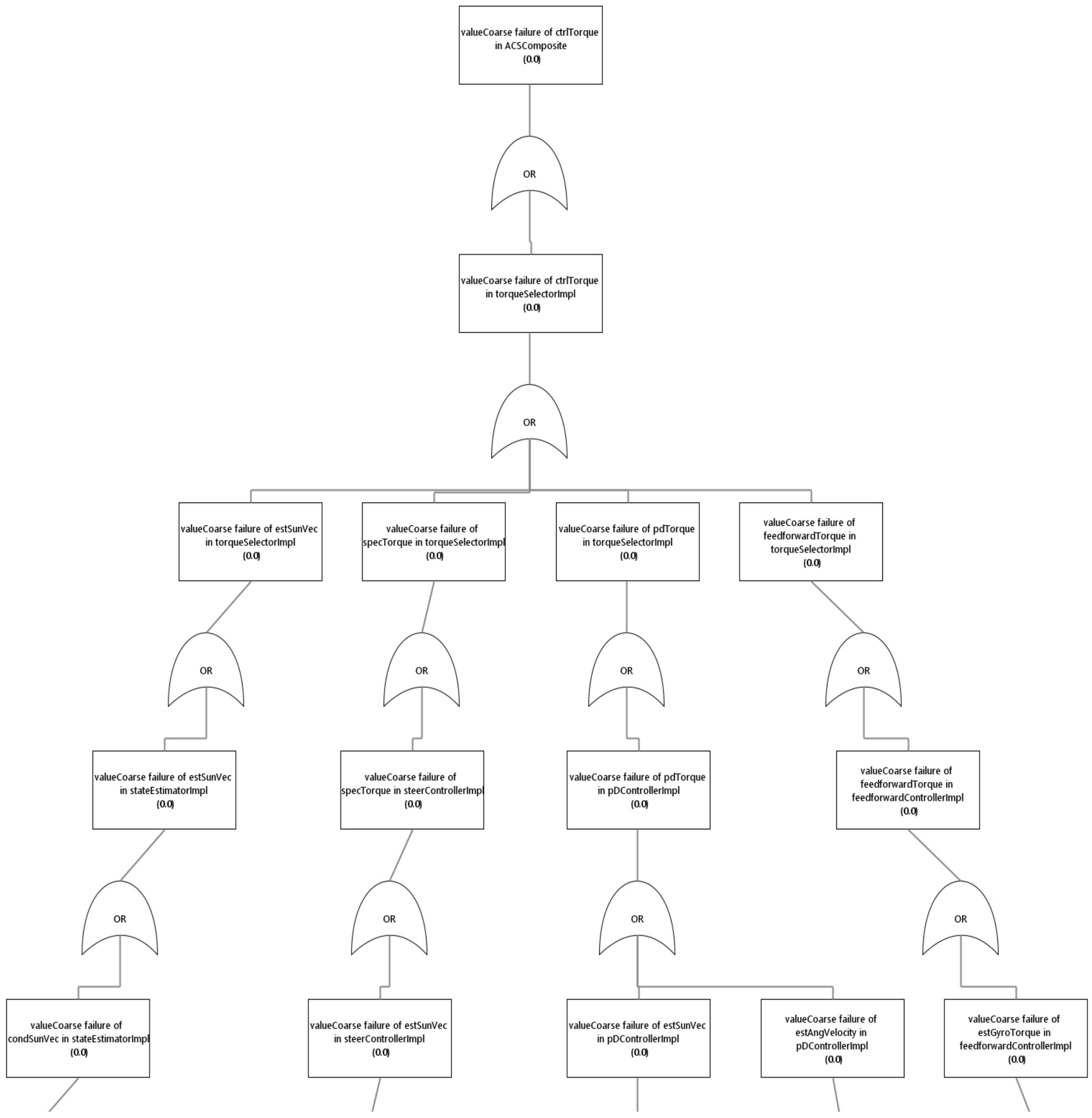


Figure 69. SignalConditioner Component Security Attack Model

ECSS Standard for product assurance has dedicated documents for dependability and safety i.e., ECSS-Q-ST-30C [52] and ECSS-Q-ST-40C [53] respectively. The former puts an emphasis on the reliability attribute of the dependability, where the latter targets safety explicitly. Both documents require the reliability and safety analysis at all the stages of product design and development. The standards suggest various analysis methods both top-down and bottom-up approaches, where fault tree analysis is one of the methods. In addition to this, a recently introduced standard ESSB-ST-E-008 [71] provides requirements of cyber security risk assessment analysis at all levels of development of space missions.

The resulting failure propagation paths stored in FLAMM file are utilized to generate multi-concern fault tree. Figure 70 shows the complete fault tree for the ACS composite component for a system level failure as “valueCoarse failure of ctrlTorque in ACSComposite” with the highlighted area corresponding to the “SignalConditioner” component”. This highlighted part of the fault tree is shown in Figure 71 , which shows both (safety and security) causes for the top-level event to occur. It shows that the “unauthorizedModificationOfService” or a valueCoarse failure at port “condSunVec” of “SignalConditioner” component can cause the “valueCoarse” failure on the “condSunVec” input port of “StateEstimator” component. The “unauthorizedModificationofService” threat is caused due to the combination of “dataSpoofingAttack” on the input port of “SignalConditioner” component and “missingDataIntegritySchemes” vulnerability.


valueCoarse failure of ctrlTorque in ACSComposite (0.0)
OR
valueCoarse failure of ctrlTorque in torqueSelectorImpl (0.0)
OR
valueCoarse failure of estSunVec in torqueSelectorImpl (0.0)
valueCoarse failure of specTorque in torqueSelectorImpl (0.0)
valueCoarse failure of pdTorque in torqueSelectorImpl (0.0)
valueCoarse failure of feedforwardTorque in torqueSelectorImpl (0.0)
OR
OR
OR
OR
valueCoarse failure of estSunVec in stateEstimatorImpl (0.0)
valueCoarse failure of specTorque in steerControllerImpl (0.0)
valueCoarse failure of pdTorque in pDControllerImpl (0.0)
valueCoarse failure of feedforwardTorque in feedforwardControllerImpl (0.0)
OR
OR
OR
OR
valueCoarse failure of condSunVec in stateEstimatorImpl (0.0)
valueCoarse failure of estSunVec in steerControllerImpl (0.0)
valueCoarse failure of estSunVec in pDControllerImpl (0.0)
valueCoarse failure of estAngVelocity in pDControllerImpl (0.0)
valueCoarse failure of estGyroTorque in feedforwardControllerImpl (0.0)

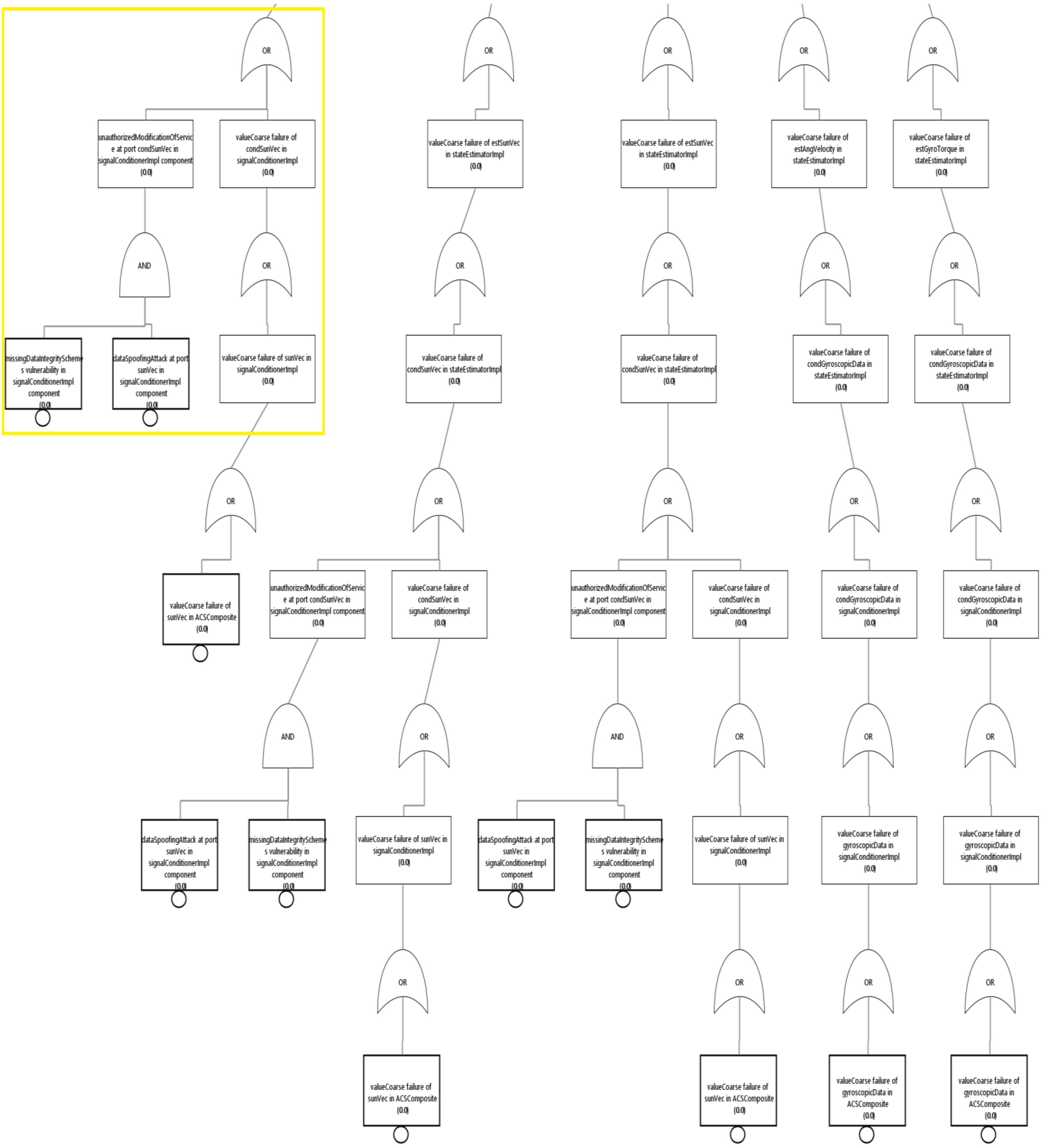

OR
unauthorizedModificationOfService at port condSunVec in signalConditionerImpl component (0.0)
valueCoarse failure of condSunVec in signalConditionerImpl (0.0)
AND
missingDataIntegritySchemes vulnerability in signalConditionerImpl component (0.0)
dataSpoofingAttack at port sunVec in signalConditionerImpl component (0.0)
valueCoarse failure of sunVec in signalConditionerImpl (0.0)
valueCoarse failure of sunVec in ACSComposite (0.0)
valueCoarse failure of estSunVec in stateEstimatorImpl (0.0)
valueCoarse failure of condSunVec in stateEstimatorImpl (0.0)
valueCoarse failure of estAngVelocity in stateEstimatorImpl (0.0)
valueCoarse failure of estGyroTorque in stateEstimatorImpl (0.0)
valueCoarse failure of condGyroscopicData in stateEstimatorImpl (0.0)
valueCoarse failure of condGyroscopicData in signalConditionerImpl (0.0)
valueCoarse failure of gyroscopicData in signalConditionerImpl (0.0)
valueCoarse failure of gyroscopicData in ACSComposite (0.0)

Figure 70. Automatically generated fault tree from failure propagation paths with highlighted SignalConditioner Component tree

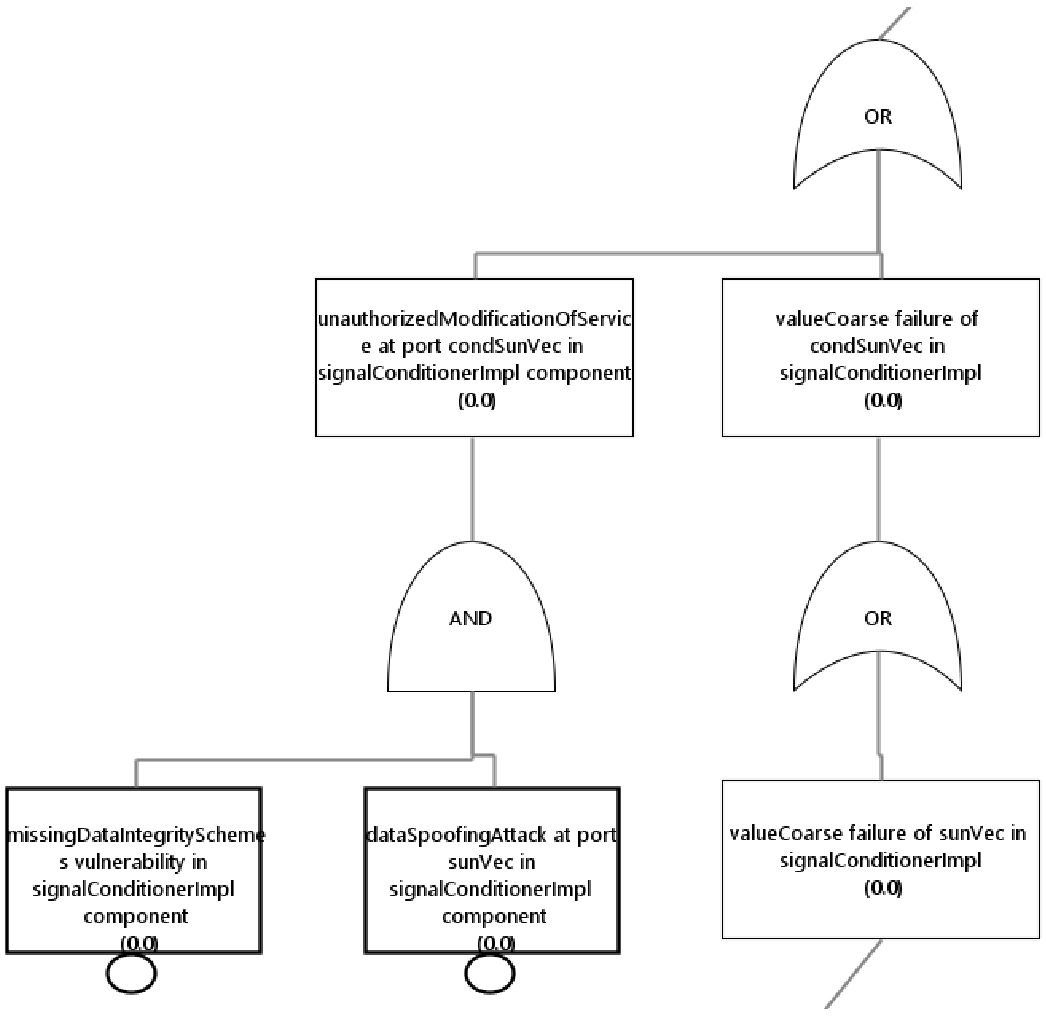

OR
unauthorizedModificationOfService at port condSunVec in signalConditionerImpl component (0.0)
valueCoarse failure of condSunVec in signalConditionerImpl (0.0)
AND
OR
missingDataIntegritySchemes vulnerability in signalConditionerImpl component (0.0)
dataSpoofingAttack at port sunVec in signalConditionerImpl component (0.0)
valueCoarse failure of sunVec in signalConditionerImpl (0.0)

Figure 71. SignalConditioner Component fault tree illustrating multi-concern causes

4.2. Case Study CS3 - Cooperative Adaptive Cruise Control (CACC)

In this section, a case study related to CS3 Automotive Case Study: Collaborative automated fleet of vehicles described in D1.1 [1] is considered. First a specific scenario is explained and then the methodological guide is applied on it.

4.2.1. Description of the Use Case Scenario

This scenario is an excerpt of CS3 Automotive Case Study: Collaborative automated fleet of vehicles described in D1.1 [1]. This Case Study handles a typical example of a collaborative safety-critical system: a platoon of several vehicles.

In this subsection, a partial argument for CACC is presented. More specifically, the argument focuses on the "rear collision" hazard under nominal, malfunctioning and malicious attack conditions, see Figure 72. The nominal behaviour module assures our confidence that the risk of rear collision occurring is acceptable when there is neither malfunctioning behaviour nor malicious attacks. The malfunctioning behaviour module focuses on safety in presence of failures in the system, while the malicious intent module addresses risks of rear collision due to malicious attack.

4.2.2. Demonstration of the methodology

The nominal behaviour module is assured through decomposition of requirements on assumption-guarantee contracts that deal with properties such as deceleration capability of the remote vehicles under platoon and CACC mode, the timing of the communication, the accuracy of the sensors as well as the guaranteed distance based on these properties.

To address the malfunctioning behaviour, we go through each identified failure combination and show that it is adequately addressed. For example, for value failures in the signal between the two cars (failComb2 goal) that are not clearly detectable, we rely on the accuracy of the remote sensor and the data integrity during the transmission. Since the link between two cars depends on the security of the channel (commChannel goal), we denote it with the dependency impact relationship.

Finally, in the malicious intent module we argue over all identified vulnerabilities. For example, if the communication channel is unsecure, the attacker can send wrong messages to different cars and hence cause the car2x value failures that may lead to rear collision. Hence, we encrypt the channel and secure the encryption key. The problem of using encryption in this system is that it impacts the timing of communication between the vehicles. On the one hand, using encryption conflicts with the timing contract specified in the nominal behaviour module. This conflicting relationship will make us re-work the design highlighted in red. A final Assurance Case should have all the conflicting relationships resolved. On the other hand, using encryption facilitates assuring the data integrity in the malfunctioning behaviour module, besides the implemented checksum. Hence, we use the choice symbol to depict that the data integrity is adequately addressed by either checksum or encryption module.

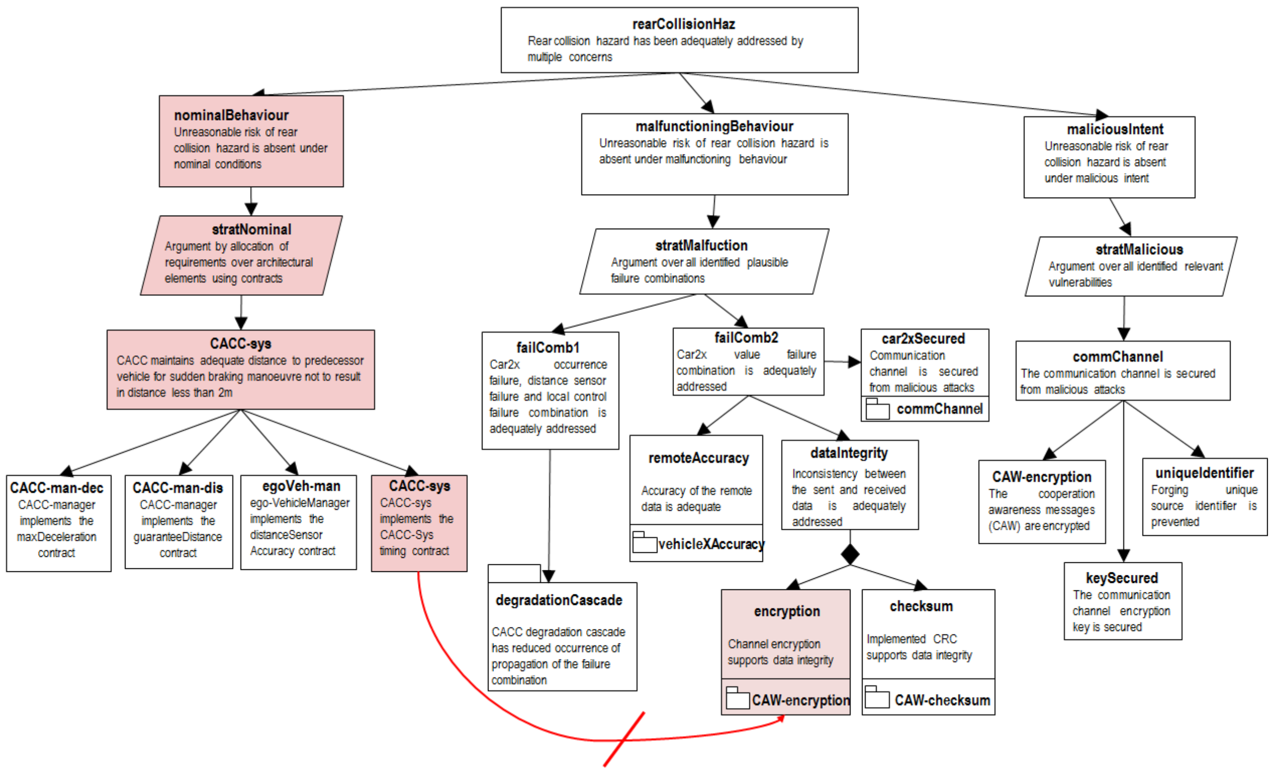

rearCollisionHaz
Rear collision hazard has been adequately addressed by multiple concerns
nominalBehaviour
Unreasonable risk of rear collision hazard is absent under nominal conditions
malfunctioningBehaviour
Unreasonable risk of rear collision hazard is absent under malfunctioning behaviour
maliciousIntent
Unreasonable risk of rear collision hazard is absent under malicious intent
stratNominal
Argument by allocation of requirements over architectural elements using contracts
stratMalfuction
Argument over all identified plausible failure combinations
stratMalicious
Argument over all identified relevant vulnerabilities
CACC-sys
CACC maintains adequate distance to predecessor vehicle for sudden braking manoeuvre not to result in distance less than 2m
failComb1
Car2x occurrence failure, distance sensor failure and local control failure combination is adequately addressed
failComb2
Car2x value failure combination is adequately addressed
car2xSecured
Communication channel is secured from malicious attacks
commChannel
commChannel
The communication channel is secured from malicious attacks
CACC-man-dec
CACC-manager implements the maxDeceleration contract
CACC-man-dis
CACC-manager implements the guaranteeDistance contract
egoVeh-man
ego-VehicleManager implements the distanceSensor Accuracy contract
CACC-sys
CACC-sys implements the CACC-Sys timing contract
remoteAccuracy
Accuracy of the remote data is adequate
vehicleXAccuracy
dataIntegrity
Inconsistency between the sent and received data is adequately addressed
CAW-encryption
The cooperation awareness messages (CAW) are encrypted
uniqueIdentifier
Forging unique source identifier is prevented
degradationCascade
CACC degradation cascade has reduced occurrence of propagation of the failure combination
encryption
Channel encryption supports data integrity
CAW-encryption
checksum
Implemented CRC supports data integrity
CAW-checksum
keySecured
The communication channel encryption key is secured

Figure 72. Assuring “rear collision” hazard in platooning/CACC capable vehicle

4.3. Process-related Dependability Co-Assessment: An Automotive Case

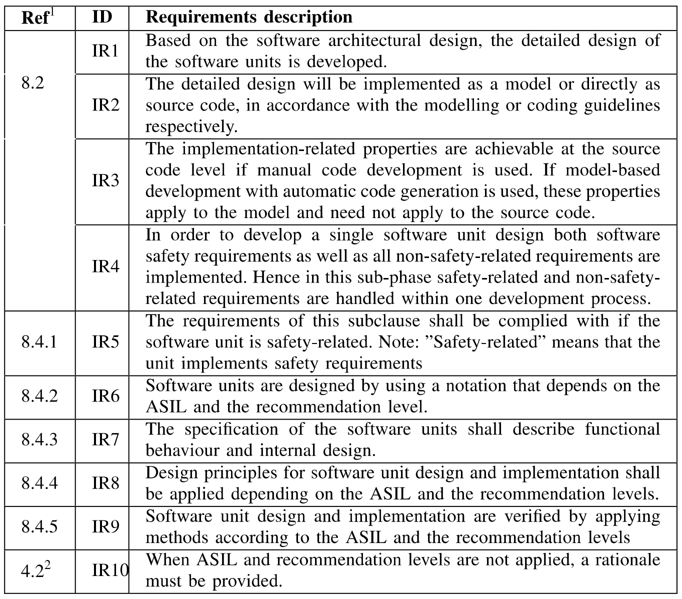
In this section, process-related safety & security co-assessment in the automotive domain is in focus. More specifically, this section builds on top of the work presented in [29] in which the alignment of ISO 26262 and SAE J3061 at the level of ‘Software Design and Implementation’ was discussed. Commonalities and variabilities in terms of work products and breaking down of the work was also discussed in [29].

In this section, first the main results presented in [29] are recalled, then the workflow, which was presented in Figure 23, is applied to co-assess safety and security at the software design and implementation level.

The following subsubsections are aimed at identifying the process elements, which can be retrieved from the normative documents, and using them within the integration of EPF Composer and BVR Tool in order to enable co-assessment.

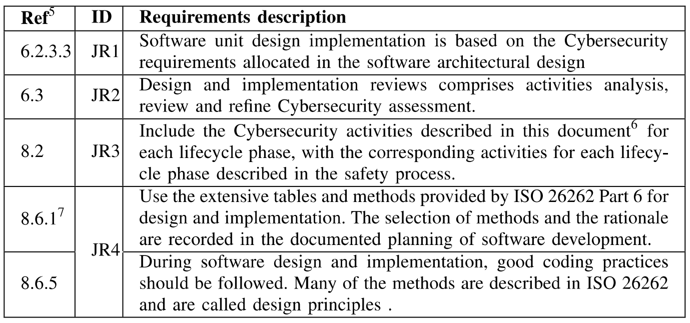
4.3.1. Commonalities and Variabilities between SAE J3061 and ISO 26262

This subsection recalls the main findings, which were presented in [29]. More specifically, the process related requirements at software design and implementation level were extrapolated (see Figure 73 and Figure 74) and compared in order to retrieve common and variation points (see Figure 75) to be exploited within the integration of EPF Composer and BVR Tool.



Ref[1]	ID	Requirements description
8.2	IR1	Based on the software architectural design, the detailed design of the software units is developed.
	IR2	The detailed design will be implemented as a model or directly as source code, in accordance with the modelling or coding guidelines respectively.
	IR3	The implementation-related properties are achievable at the source code level if manual code development is used. If model-based development with automatic code generation is used, these properties apply to the model and need not apply to the source code.
	IR4	In order to develop a single software unit design both software safety requirements as well as all non-safety-related requirements are implemented. Hence in this sub-phase safety-related and non-safety-related requirements are handled within one development process.
8.4.1	IR5	The requirements of this subclause shall be complied with if the software unit is safety-related. Note: "Safety-related" means that the unit implements safety requirements
8.4.2	IR6	Software units are designed by using a notation that depends on the ASIL and the recommendation level.
8.4.3	IR7	The specification of the software units shall describe functional behaviour and internal design.
8.4.4	IR8	Design principles for software unit design and implementation shall be applied depending on the ASIL and the recommendation levels.
8.4.5	IR9	Software unit design and implementation are verified by applying methods according to the ASIL and the recommendation levels
4.2[2]	IR10	When ASIL and recommendation levels are not applied, a rationale must be provided.

Figure 73. Requirements from ISO 26262



Ref[5]	ID	Requirements description
6.2.3.3	JR1	Software unit design implementation is based on the Cybersecurity requirements allocated in the software architectural design
6.3	JR2	Design and implementation reviews comprises activities analysis, review and refine Cybersecurity assessment.
8.2	JR3	Include the Cybersecurity activities described in this document[6] for each lifecycle phase, with the corresponding activities for each lifecycle phase described in the safety process.
8.6.1[7]	JR4	Use the extensive tables and methods provided by ISO 26262 Part 6 for design and implementation. The selection of methods and the rationale are recorded in the documented planning of software development.
8.6.5		During software design and implementation, good coding practices should be followed. Many of the methods are described in ISO 26262 and are called design principles .

Figure 74. Requirements from SAE J3061

ID	IR	JR	Common Name
CP1	IA1	JA1	Unit design
VP1a	IA1		Design concerning safety
VP1b		JA1	Design concerning cybersecurity
CP2	IA2	JA3	Unit design review
VP2a	IA2		Design review concerning safety
VP2b		JA3	Design review concerning cybersecurity
CP3	IA3	JA2	Unit implementation
VP3a	IA3		Unit implementation concerning safety
VP3b		JA2	Unit implementation concerning cybersecurity
CP4	IA4	JA4	Unit implementation review
VP4a	IA4		Implementation review concerning safety
VP4b		JA4	Implementation review concerning cybersecurity

Figure 75. Common and Variation Points identification

4.3.2. Work Products

Based on the information contained in the two normative documents, the following work products are defined in our case study:

- Software Unit Design Specification
- Software Unit Implementation
- Software Safety Requirements
- Software Cybersecurity Requirements

Software Unit Design Specification, Software Safety Requirements and Software Cybersecurity Requirements serve as inputs to the various design tasks. These inputs have been created in another phase of the software development process, which is outside the scope of the current use case. Software Unit Implementation is an output of the design tasks and also serves as input to the review and implementation tasks.

4.3.3. Roles

Based on the experience achieved in the context of critical system engineering, the following roles are defined in our case study. These roles (played by engineers with different expertise) perform the design, implementation and review tasks:

- Software Designer (assumption regarding expertise: non-safety and non-security)
- Safety Engineer (assumption: safety expert)
- Security Engineer (assumption: cybersecurity expert)
- Programmer, i.e., engineer with programming expertise (assumption regarding expertise: non-safety and non-security)
- Software Tester, i.e., engineer with testing expertise

The Software Designer, the Safety Engineer and the Security Engineer are responsible for design tasks (see 4.3.5) and are also responsible for the Software Unit Implementation work product. The Programmer is responsible for all implementation tasks while the Software Tester is responsible for review tasks, both design review as well as implementation review.

4.3.4. Guidance

Based on what is stated in the normative documents, various Guidance elements relevant to the various tasks can be defined. These elements are listed below:

- Design Notations
- Rationale about Design Notations
- Design Principles
- Rationale about Design Principles
- Verification Methods
- Rationale about Verification Methods
- Design Activities Analysis
- Design Assessment Refinement
- Modelling Guidelines
- Source Code Guidelines
- Implementation Verification Methods
- Rationale about Implementation Verification Methods
- Implementation Verification Methods (Safety-related)
- Cybersecurity Implementation Activities Analysis
- Cybersecurity Implementation Assessment Refinement

Design of software units is done using Design Notations and Design Principles which are selected according to ASIL and recommendation levels for safety-related design. They are supported by Rationale about Design Notations and Rationale about Design Principles respectively. Software unit design verification is done using Verification Methods. Verification Methods for safety design are selected according to ASIL and recommendation levels. Verification Methods are also supported by Rationale about Verification Methods. For cybersecurity-related design, software unit design has Design Activities Analysis and Design Assessment Refinement. Software implementation is either done as a model or as source code which has Modelling Guidelines and Source Code Guidelines respectively. Software implementation verification is done using Implementation Verification Methods and supported by Rationale about Implementation Verification Methods. Safety-related Implementation Verification Methods are selected according to ASIL and recommendation levels. Cybersecurity-related implementation verification, Cybersecurity Implementation Activities Analysis and Cybersecurity Implementation Assessment Refinement are used.

4.3.5. Tasks

The following tasks related to ISO 26262 and SAE J3061 are defined. The defined tasks are related to the Work Products, Roles and Guidance defined in Sections 4.3.1, 4.3.3 and 4.3.4. These relationships are depicted in Table 1.

Table 1. Task/Work Product/Roles/Guidance Relationships

Task	Work Products	Roles	Guidance
Unit Design (Common)	(Input) Software Unit Design Specification. (Output) Software Unit Implementation.	Software Designer.	Design Notations. Rationale about Design Notations. Design Principles. Rationale about Design Principles.
Design Concerning Safety (ISO 26262)	(Input) Software Unit Design Specification. (Input) Software Safety Requirements. (Output) Software Unit Implementation.	(Primary) Safety Engineer. (Additional) Software Designer.	Design Notations. Rationale about Design Notations. Design Principles. Rationale about Design Principles.
Design Concerning Cybersecurity	(Input) Software Unit Design Specification. (Input) Software	(Primary) Security Engineer. (Additional)	Design Notations. Rationale about Design Notations.

Task	Work Products	Roles	Guidance
(SAE J3061)	Cybersecurity Requirements. (Output) Software Unit Implementation.	Software Designer.	Design Principles. Rationale about Design Principles.
Unit Design Review (Common)	(Input) Software Unit Design Specification. (Input) Software Unit Implementation.	(Primary) Software Tester. (Additional) Software Designer.	Verification Methods. Rationale about Verification Methods.
Design Review Concerning Safety (ISO 26262)	(Input) Software Unit Design Specification. (Input) Software Unit Implementation.	(Primary) Software Tester. (Additional) Software Designer. (Additional) Safety Engineer.	Verification Methods. Rationale about Verification Methods.
Design Review Concerning Cybersecurity (SAE J3061)	(Input) Software Unit Design Specification. (Input) Software Unit Implementation.	(Primary) Software Tester. (Additional) Software Designer. (Additional) Security Engineer.	Design Activities Analysis. Design Assessment Refinement.
Unit Implementation (Common)	(Optional Input) Software Unit Design Specification. (Input) Software Unit Implementation.	Programmer.	Modelling Guidelines. Source Code Guidelines.
Implementation Concerning Safety (ISO 26262)	(Optional Input) Software Unit Design Specification. (Input) Software Unit Implementation.	Programmer.	Modelling Guidelines. Source Code Guidelines.
Implementation Concerning Cybersecurity (SAE J3061)	(Optional Input) Software Unit Design Specification. (Input) Software Unit Implementation.	Programmer.	Modelling Guidelines. Source Code Guidelines.
Unit Implementation Review (Common)	(Optional Input) Software Unit Design Specification. (Input) Software Unit Implementation.	(Primary) Software Tester. (Additional) Software Designer.	Implementation Verification Methods. Rationale about Implementation Verification Methods.
Implementation Review Concerning Safety (ISO 26262)	(Optional Input) Software Unit Design Specification. (Input) Software Unit Implementation.	(Primary) Software Tester. (Additional) Software Designer. (Additional) Safety Engineer.	Implementation Verification Methods (Safety-related)
Implementation Review Concerning Cybersecurity (SAE J3061)	(Optional Input) Software Unit Design Specification. (Input) Software Unit Implementation.	(Primary) Software Tester. (Additional) Software Designer. (Additional) Security Engineer.	Cybersecurity Implementation Activities Analysis. Cybersecurity Implementation Assessment Refinement.

4.3.6. Work Break Down Structure

The Software Unit Design and Implementation Safety process is defined as a Work Break Down Structure (WBS). The WBS is composed of phases (delivery milestones), iterations, activities, tasks and their respective precedence rules. The defined WBS is shown in Table 2. The work products and team allocation are also defined for the tasks in the WBS.

Table 2. Work Break Down Structure

Index	Process/Phase/Activity/Task	Type	Predecessors
0	Software Unit Design and Implementation	Delivery Process	
1	Design And Design Review (One)	Phase	
2	Unit Design And Review – Commonality	Iteration	
3	Unit Design – Commonality	Activity	
4	Unit Design	Task	
5	Unit Design Review – Commonality	Activity	3
6	Unit Design Review	Task	
7	Implementation And Implementation Review (One)	Phase	1
8	Unit Implementation And Review – Commonality	Iteration	
9	Unit Implementation – Commonality	Activity	
10	Unit Implementation	Task	
11	Unit Implementation Review – Commonality	Activity	9
12	Unit Implementation Review	Task	
13	Design And Design Review (Two)	Phase	7
14	Unit Design And Review – Variability	Iteration	
15	Unit Design – Variability	Activity	
16	Design Concerning Safety	Task	
17	Unit Design Review – Variability	Activity	15
18	Design Review Concerning Safety	Task	
19	Implementation And Implementation Review (Two)	Phase	13
20	Unit Implementation And Review – Variability	Iteration	
21	Unit Implementation – Variability	Activity	
22	Implementation Concerning Safety	Task	
23	Unit Implementation Review - Variability	Activity	21
24	Implementation Review Concerning Safety	Task	

4.3.7. Domain Engineering

During the domain engineering phase, which is aimed at generating a Security-informed Safety-oriented Process Line (SoPL), all process elements, identified in the previous subsubsections, are interpreted as features (listed in Table 3). These features can be mandatory for all concerns or can be selected only if needed (i.e., optional feature).

Constraints among features are also specified during this phase. The constraints are specified to enforce rules for valid combinations of feature selection. To minimize the number of complex constraints, multiple simple constraints are specified instead and associated to a feature.

Table 3. Feature Tree – Variability Model

Feature	Mandatory/ Optional	Cardinality	Reference
Software Unit Design and Implementation	-	-	
ProcessModel	-	-	
ConcernChoice	Mandatory	1..1	

Feature	Mandatory/ Optional	Cardinality	Reference
Safety	Optional	-	
Cybersecurity	Optional	-	
MultiConcern	Optional	-	
Activities	Mandatory	-	**Section 4.3.5**
Commonality Point	Mandatory	-	
DesignCom	Mandatory	-	
DesignReviewCom	Mandatory	-	
ImplementationCom	Mandatory	-	
ImplementationReviewCom	Mandatory	-	
Variability Point	Mandatory	-	
DesignVar	Mandatory	1..1	
DesignSafety	Optional	-	
DesignCybersecurity	Optional	-	
DesignMultiConcern	Optional	-	
DesignSafety	Mandatory	-	
DesignCybersecurity	Mandatory	-	
DesignReviewVar	Mandatory	1..1	
DesignReviewSafety	Optional	-	
DesignReviewCybersecurity	Optional	-	
DesignReviewMultiConcern	Optional	-	
DesignReviewSafety	Mandatory	-	
DesignReviewCybersecurity	Mandatory	-	
ImplementationVar	Mandatory	1..1	
ImplementationSafety	Optional	-	
ImplementationCybersecurity	Optional	-	
ImplementationMultiConcern	Optional	-	
ImplementationSafety	Mandatory	-	
ImplementationCybersecurity	Mandatory	-	
ImplementationReviewVar	Mandatory	1..1	
ImplementationReviewSafety	Optional	-	
ImplementationReviewCybersecurity	Optional	-	
ImplementationReviewMultiConcern	Optional	-	
ImplementationReviewSafety	Mandatory	-	
ImplementationReviewCybersecurity	Mandatory	-	
Work Products	Mandatory	-	**Section 4.3.2**
SoftwareUnitDesignSpecification	Optional	-	
SoftwareUnitImplementation	Optional	-	
SoftwareSafetyRequirements	Optional	-	
SoftwareCybersecurityRequirements	Optional	-	
Guidance	Mandatory	-	**Section 4.3.4**
DesignNotations	Optional	-	
RationaleDesignNotations	Optional	-	
DesignPrinciples	Optional	-	
RationaleDesignPrinciples	Optional	-	
VerificationMethods	Optional	-	
RationaleVerificationMethods	Optional	-	
DesignActivitiesAnalysis	Optional	-	
DesignAssessmentRefinement	Optional	-	

Feature	Mandatory/ Optional	Cardinality	Reference
ModellingGuidelines	Optional	-	
SourceCodeGuidelines	Optional	-	
ImplementationVerificationMethods	Optional	-	
RationaleImplementationVerificationMethods	Optional	-	
ImplementationVerificationMethodsSafety	Optional	-	
CybersecurityImplementationActivitiesAnalysis	Optional	-	
CybersecurityImplementationAssessmentRefinement	Optional	-	
Roles	Mandatory	1..*	**Section 4.3.3**
SoftwareDesigner	Optional	-	
SafetyEngineer	Optional	-	
Programmer	Optional	-	
SoftwareTester	Optional	-	

4.3.8. Variability Model Creation (VSpec Editor)

The features, which were engineered and presented in Table 3, are, in this subsection, organized in a tree structure, which represents the SoPL model regarding the software design and implementation. More precisely, by using the VSpec Editor in BVR, the SoPL model is created and its representation is shown in Figure 76. This model shows that a Process Model can be configured in various ways depending on the feature selection. The features that compose the Process Model are: ConcernChoice, Activities, WorkProducts, Guidance and Roles.

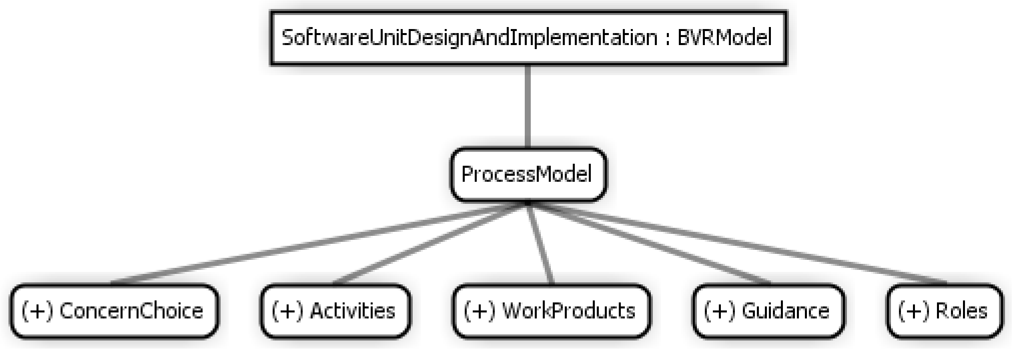


Figure 76. Feature Tree – Top Level

A step by step evolution of the SoPL model, along with the related constraints, is depicted in the following figures (Figure 77 through Figure 97).

Figure 77 depicts a view with ConcernChoice, Roles and Activities expanded. ConcernChoice is expanded with three choices, Safety, Cybersecurity and MultiConcern, and any one and only one (cardinality 1..1) may be chosen at a time. The Roles are SoftwareDesigner, SafetyEngineer, Programmer and SoftwareTester. At least one Role must be selected for any process model. However, one may select more than one Role too (cardinality 1..*). Activities are split into two sub trees, CommonalityPoint (activities which are common to the choices of ConcernChoice) and VariabilityPoint (activities which differ for each one of the choices of ConcernChoice).

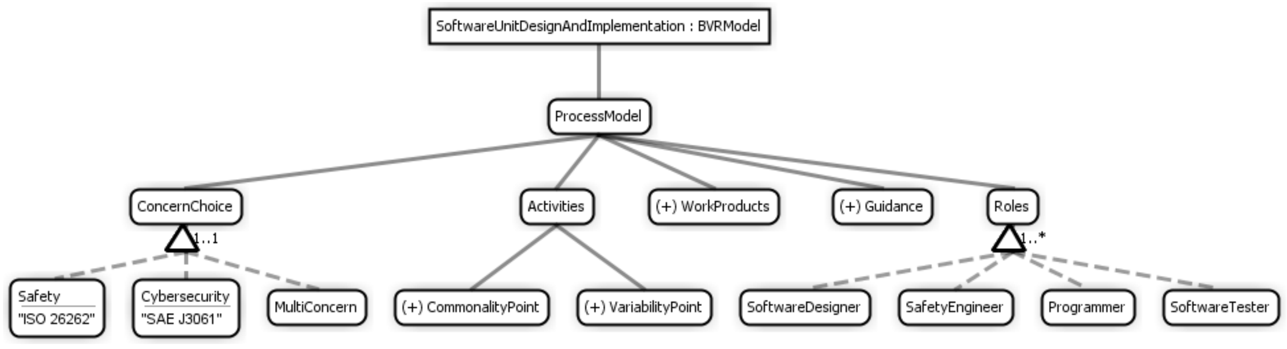


Figure 77. ConcernChoice, Roles and Activities Expanded

The WorkProducts subtree is depicted in Figure 78 along with the relevant constraints. The WorkProducts are SoftwareUnitDesignSpecification, SoftwareUnitImplementation, SoftwareSafetyRequirements and SoftwareCybersecurityRequirements. WorkProducts are optionally chosen for a Process Model. The constraints specify the need for SoftwareSafetyRequirements in the case ConcernChoice is Safety or MultiConcern and the need for SoftwareCybersecurityRequirements in the case ConcernChoice is Cybersecurity or MultiConcern.

The Guidance sub tree is depicted in Figure 79 and Figure 80. The sub tree has been split into two figures to ensure readability. The relevant constraints are also shown. For example, RationaleDesignNotations implies the existence of DesignNotations.

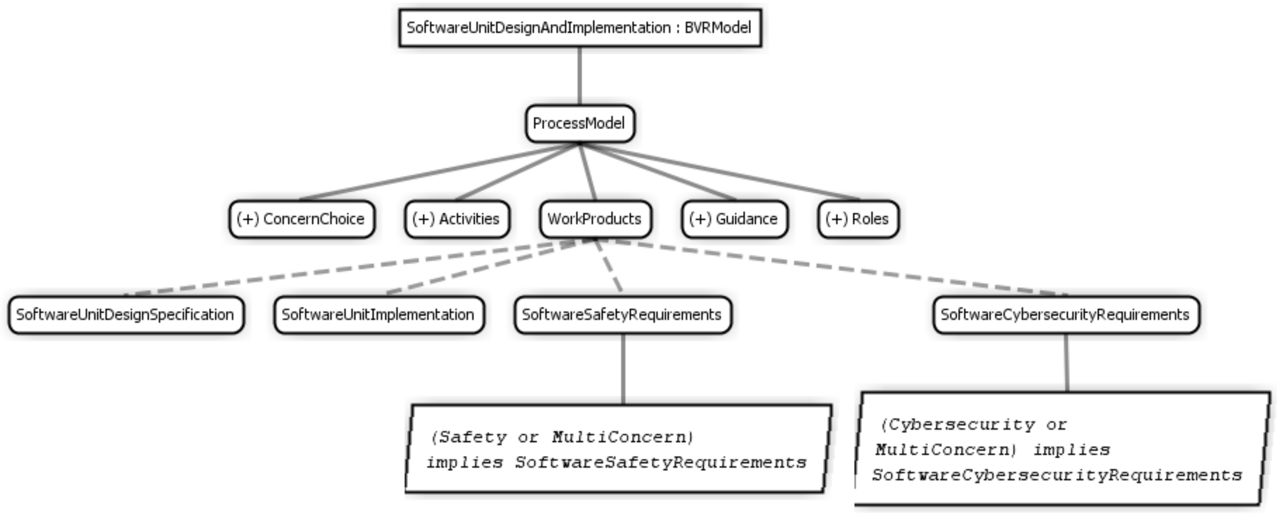

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
(+) Activities
WorkProducts
(+) Guidance
(+) Roles
SoftwareUnitDesignSpecification
SoftwareUnitImplementation
SoftwareSafetyRequirements
SoftwareCybersecurityRequirements
(Safety or MultiConcern) implies SoftwareSafetyRequirements
(Cybersecurity or MultiConcern) implies SoftwareCybersecurityRequirements

Figure 78. WorkProducts Subtree

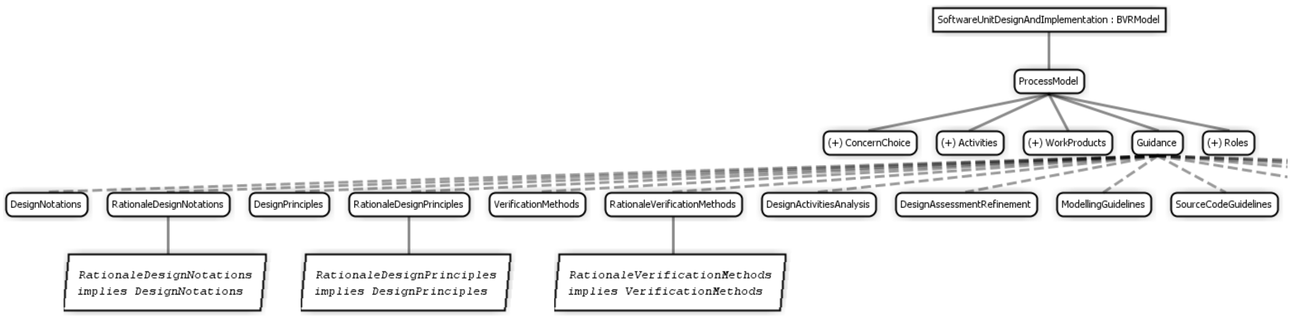

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
(+) Activities
(+) WorkProducts
Guidance
(+) Roles
DesignNotations
RationaleDesignNotations
DesignPrinciples
RationaleDesignPrinciples
VerificationMethods
RationaleVerificationMethods
DesignActivitiesAnalysis
DesignAssessmentRefinement
ModellingGuidelines
SourceCodeGuidelines
RationaleDesignNotations implies DesignNotations
RationaleDesignPrinciples implies DesignPrinciples
RationaleVerificationMethods implies VerificationMethods

Figure 79. Guidance Subtree (1 of 2)

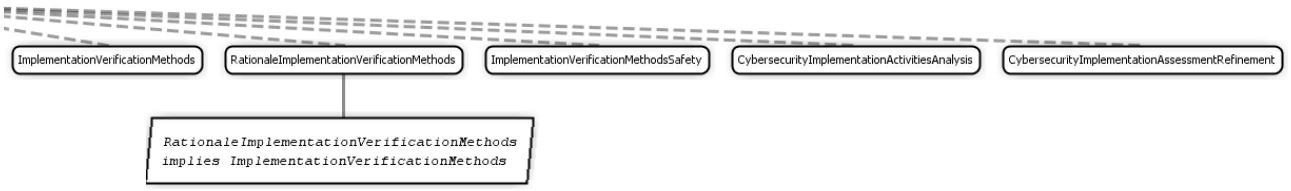

ImplementationVerificationMethods
RationaleImplementationVerificationMethods
ImplementationVerificationMethodsSafety
CybersecurityImplementationActivitiesAnalysis
CybersecurityImplementationAssessmentRefinement
RationaleImplementationVerificationMethods implies ImplementationVerificationMethods

Figure 80. Guidance Subtree (2 of 2)

The CommonalityPoint and VariabilityPoint subtrees are depicted in Figure 81. The CommonalityPoint subtree consists of activities, which are common to all choices of ConcernChoice. These common activities are: DesignCom, DesignReviewCom, ImplementationCom, and ImplementationReviewCom.

The VariabilityPoint subtree consists of activities, which contribute to discriminate the concerns. These activities are: DesignVar, DesignReviewVar, ImplemenationVar, and ImplementationReviewVar.

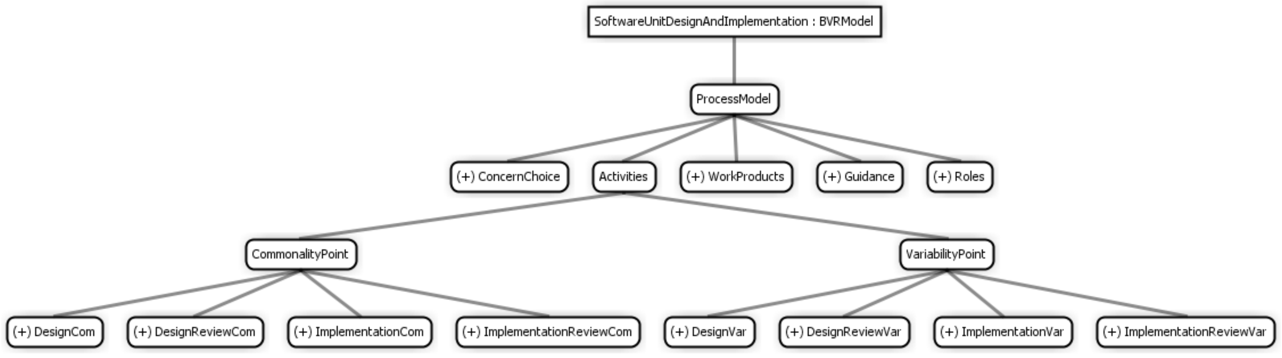


Figure 81. CommonalityPoint and VariabilityPoint Subtrees

The subtree corresponding to DesignCom is depicted in Figure 82. This subtree does not contain any further levels. There are three constraints related to Roles, Work Products and Guidance specified.

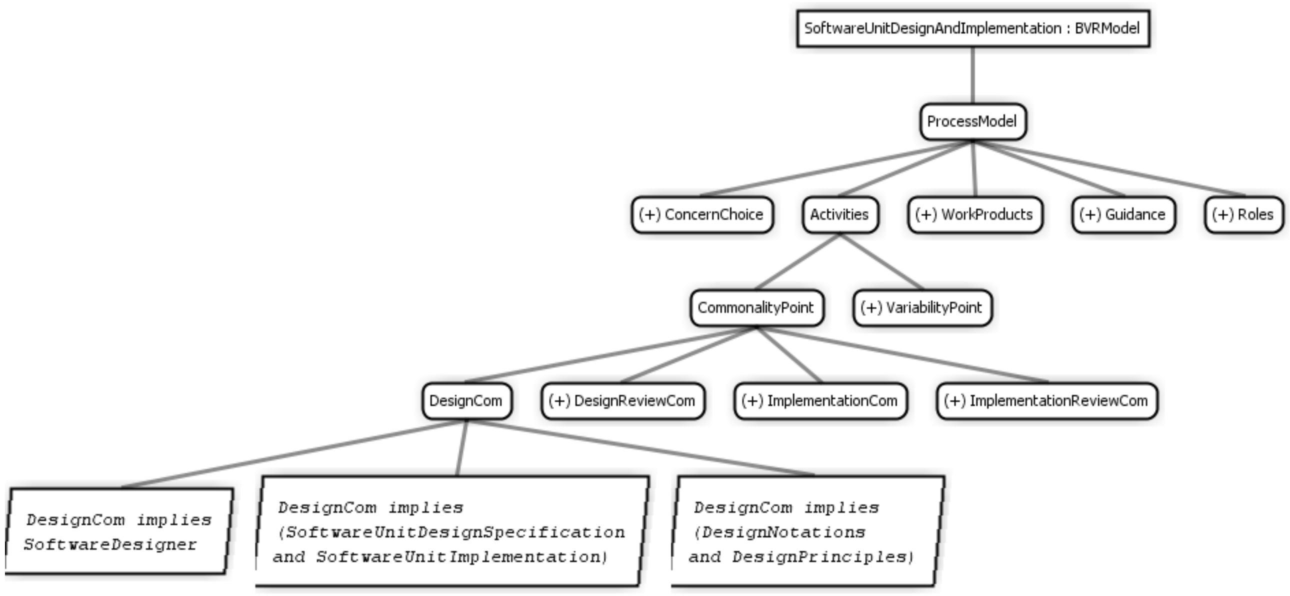

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
CommonalityPoint
(+) VariabilityPoint
DesignCom
(+) DesignReviewCom
(+) ImplementationCom
(+) ImplementationReviewCom
DesignCom implies SoftwareDesigner
DesignCom implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
DesignCom implies (DesignNotations and DesignPrinciples)

Figure 82. DesignCom Subtree

The subtree corresponding to DesignReviewCom is depicted in Figure 83. This subtree does not contain any further levels. There are three constraints related to Roles, Work Products and Guidance specified.

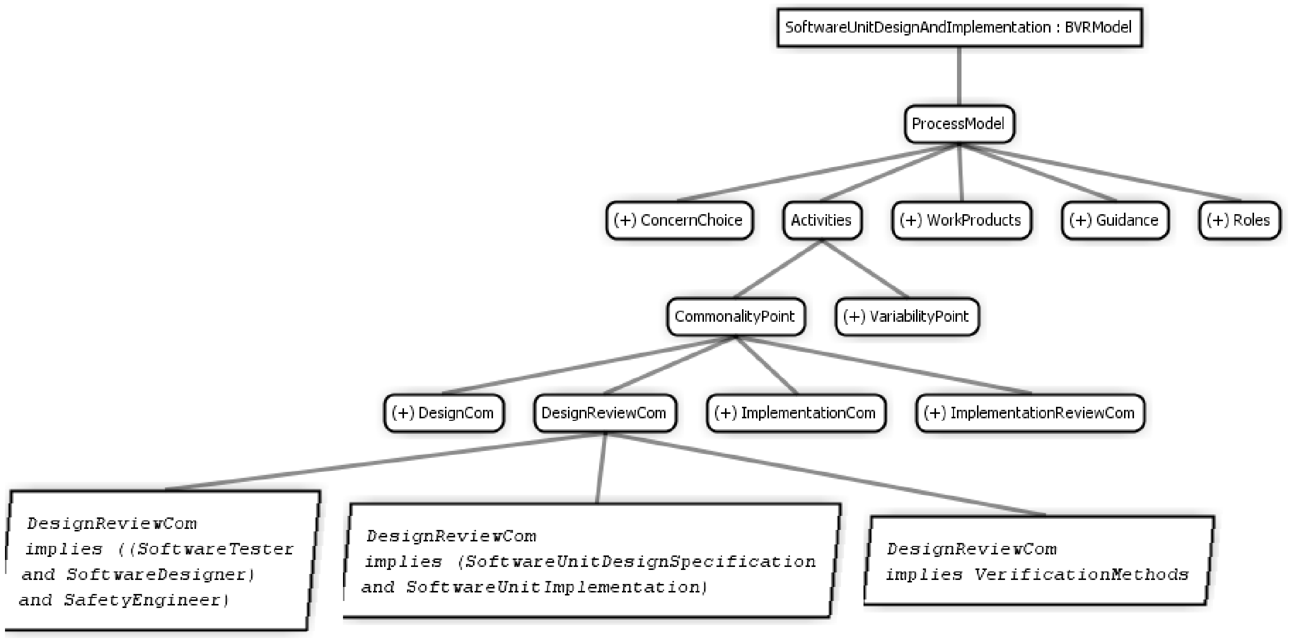

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
CommonalityPoint
(+) VariabilityPoint
(+) DesignCom
DesignReviewCom
(+) ImplementationCom
(+) ImplementationReviewCom
DesignReviewCom implies ((SoftwareTester and SoftwareDesigner) and SafetyEngineer)
DesignReviewCom implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
DesignReviewCom implies VerificationMethods

Figure 83. DesignReviewCom Subtree

The subtree corresponding to ImplementationCom is depicted in Figure 84. This subtree does not contain any further levels. There are three constraints related to Roles, Work Products and Guidance specified.

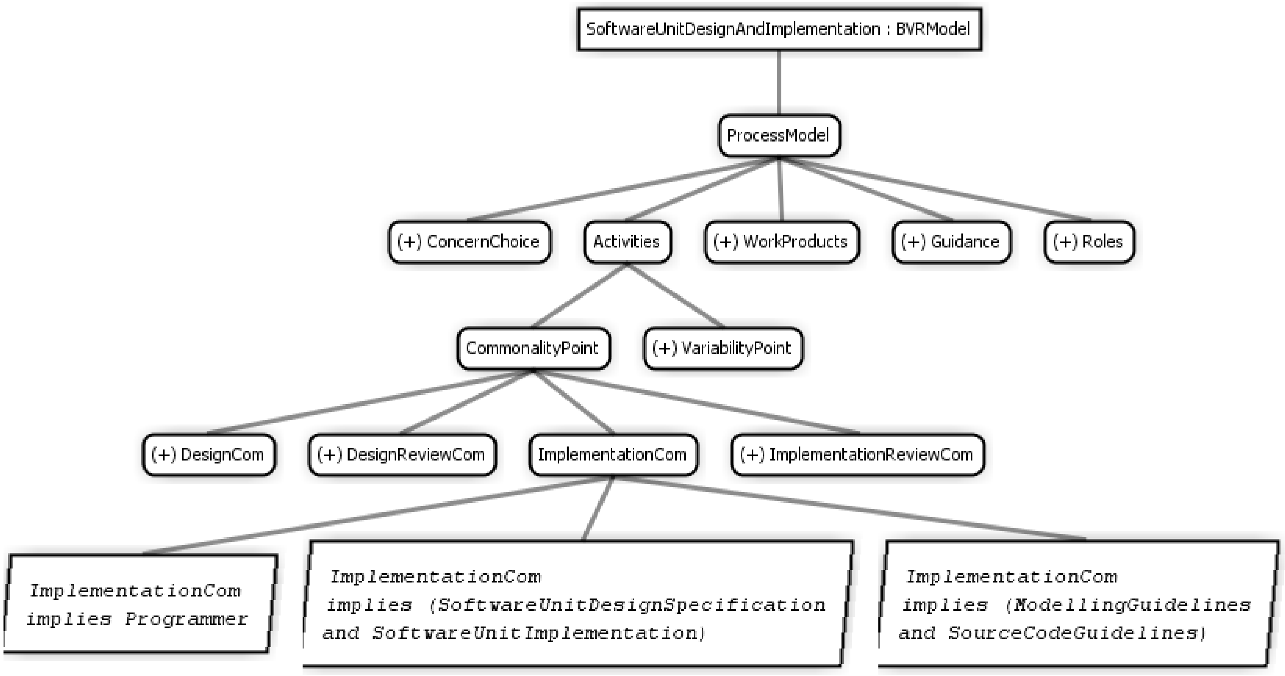

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
CommonalityPoint
(+) VariabilityPoint
(+) DesignCom
(+) DesignReviewCom
ImplementationCom
(+) ImplementationReviewCom
ImplementationCom implies Programmer
ImplementationCom implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
ImplementationCom implies (ModellingGuidelines and SourceCodeGuidelines)

Figure 84. ImplementationCom Subtree

The subtree corresponding to ImplementationReviewCom is depicted in Figure 85. This subtree does not contain any further levels. There are three constraints related to Roles, Work Products and Guidance specified.

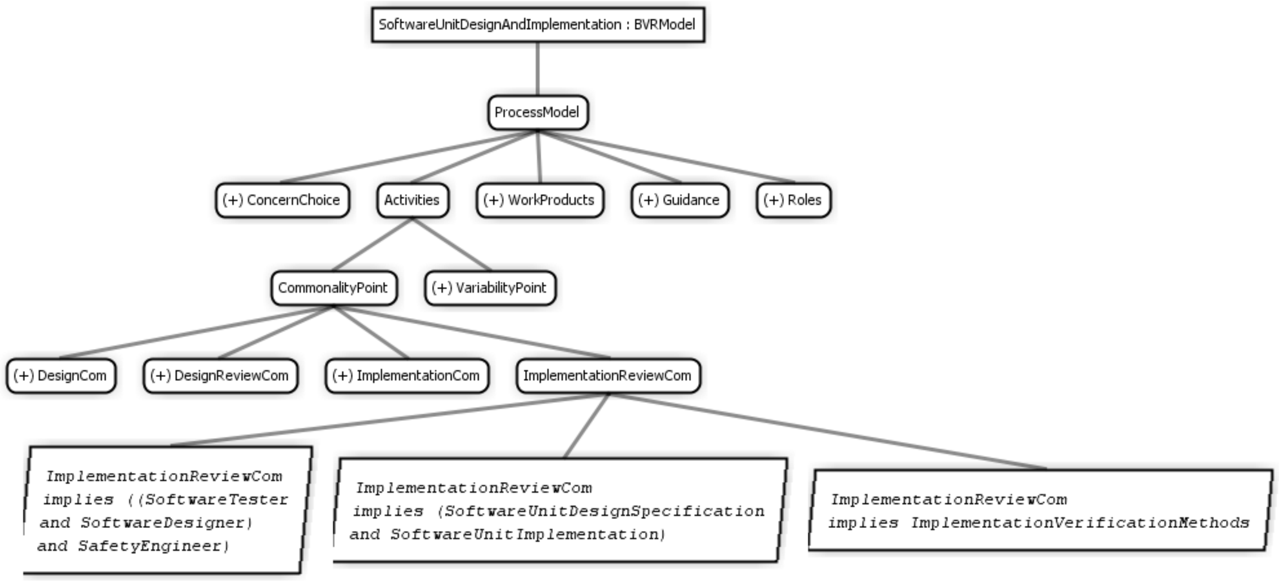

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
CommonalityPoint
(+) VariabilityPoint
(+) DesignCom
(+) DesignReviewCom
(+) ImplementationCom
ImplementationReviewCom
ImplementationReviewCom implies ((SoftwareTester and SoftwareDesigner) and SafetyEngineer)
ImplementationReviewCom implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
ImplementationReviewCom implies ImplementationVerificationMethods

Figure 85. ImplementationReviewCom Subtree

The DesignVar and DesignSafety subtrees are depicted in Figure 86. The DesignVar subtree consists of three elements, DesignSafety, DesignCybersecurity and DesignMultiConcern, related to each one of the choices of ConcernChoice and connected in exclusive-OR manner. The constraint restricting DesignSafety only to a choice of Safety for ConcernChoice ensures the exclusive-OR relation. Besides, DesignSafety also consists of constraints for Roles, WorkProducts and Guidance. DesignSafety does not have any further level.

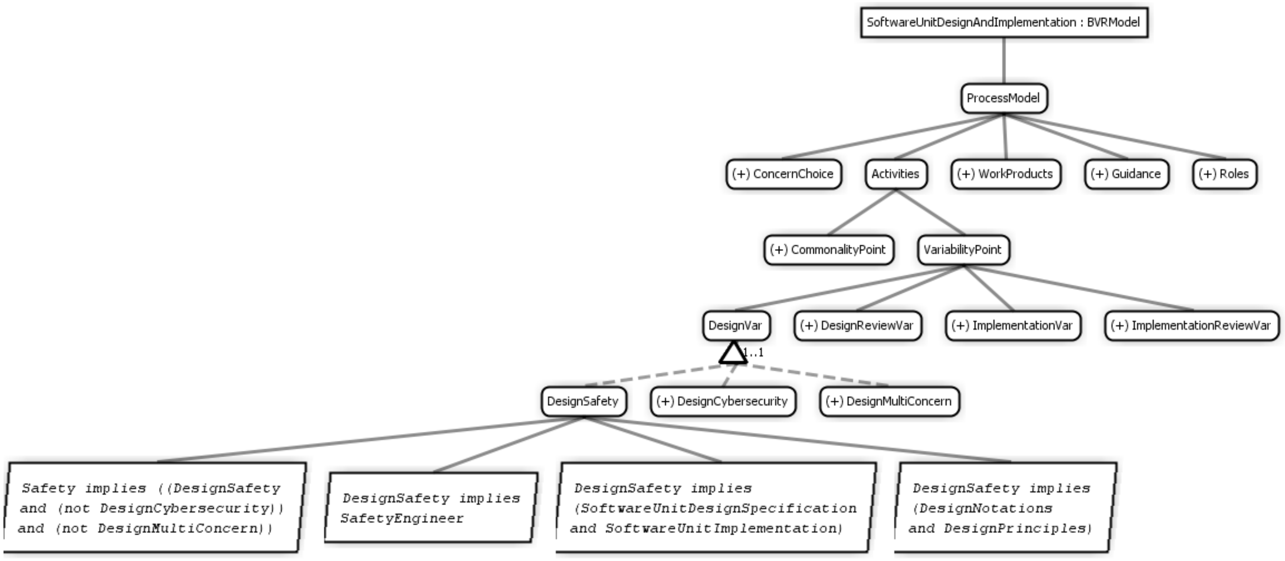

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
DesignVar
(+) DesignReviewVar
(+) ImplementationVar
(+) ImplementationReviewVar
1..1
DesignSafety
(+) DesignCybersecurity
(+) DesignMultiConcern
Safety implies ((DesignSafety and (not DesignCybersecurity)) and (not DesignMultiConcern))
DesignSafety implies SafetyEngineer
DesignSafety implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
DesignSafety implies (DesignNotations and DesignPrinciples)

Figure 86. DesignVar and DesignSafety Subtrees

The DesignVar and DesignCybersecurity subtrees are depicted in Figure 87. The constraint restricting DesignCybersecurity only to a choice of Cybersecurity for ConcernChoice ensures the exclusive-OR relation. Besides, DesignCybersecurity also consists of constraints for Roles, WorkProducts and Guidance. DesignCybersecurity does not have any further level.

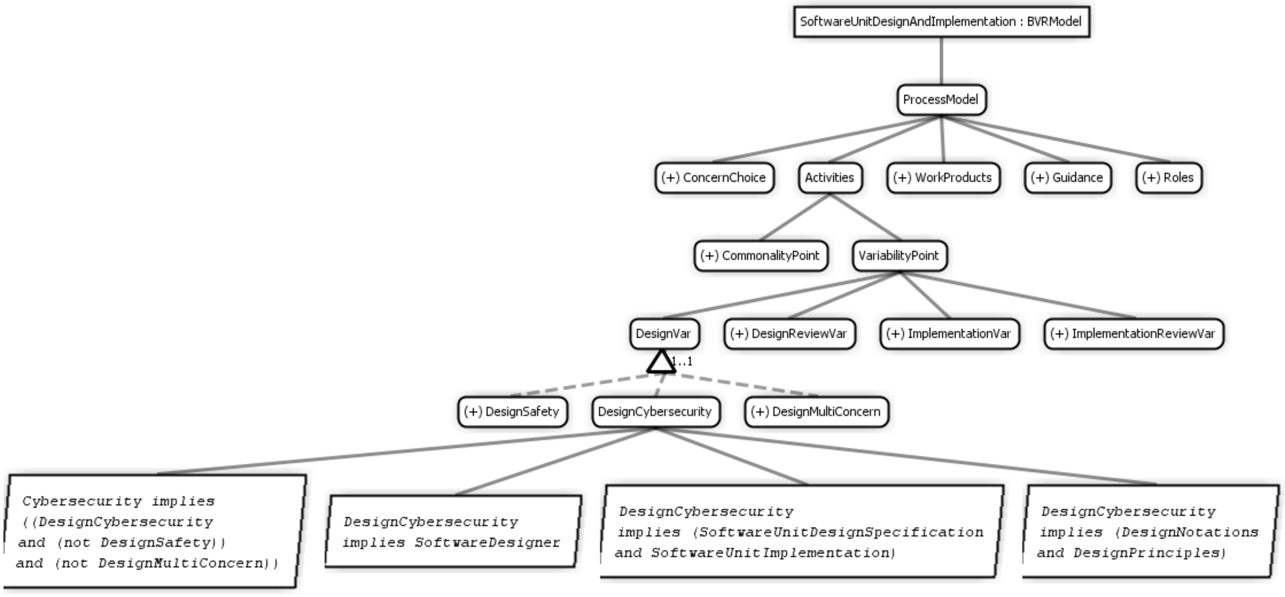

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
DesignVar
(+) DesignReviewVar
(+) ImplementationVar
(+) ImplementationReviewVar
1..1
(+) DesignSafety
DesignCybersecurity
(+) DesignMultiConcern
Cybersecurity implies ((DesignCybersecurity and (not DesignSafety)) and (not DesignMultiConcern))
DesignCybersecurity implies SoftwareDesigner
DesignCybersecurity implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
DesignCybersecurity implies (DesignNotations and DesignPrinciples)

Figure 87. DesignVar and DesignCybersecurity Subtrees

The DesignVar and DesignMultiConcern subtrees are depicted in Figure 88. The constraint restricting DesignMultiConcern only to a choice of MultiConcern for ConcernChoice ensures the exclusive-OR relation. Besides, DesignMultiConcern also consists of constraints for Roles, WorkProducts and Guidance. DesignMultiConcern also has a next level consisting of elements DesignSafety and DesignCybersecurity to cover multi concern design tasks.

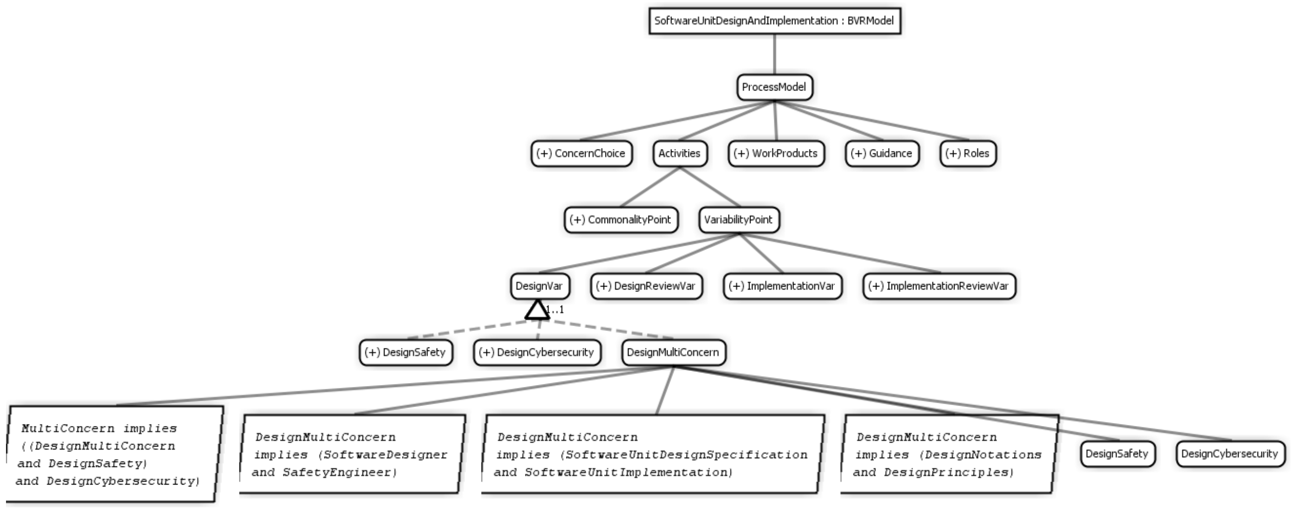

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
DesignVar
(+) DesignReviewVar
(+) ImplementationVar
(+) ImplementationReviewVar
1..1
(+) DesignSafety
(+) DesignCybersecurity
DesignMultiConcern
MultiConcern implies ((DesignMultiConcern and DesignSafety) and DesignCybersecurity)
DesignMultiConcern implies (SoftwareDesigner and SafetyEngineer)
DesignMultiConcern implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
DesignMultiConcern implies (DesignNotations and DesignPrinciples)
DesignSafety
DesignCybersecurity

Figure 88. DesignVar and DesignMultiConcern Subtrees

The DesignReviewVar and DesignReviewSafety subtrees are depicted in Figure 89. The DesignReviewVar subtree consists of three elements, DesignReviewSafety, DesignReviewCybersecurity and DesignReviewMultiConcern, related to each one of the choices of ConcernChoice and connected in exclusive-OR manner. The constraint restricting DesignReviewSafety only to a choice of Safety for ConcernChoice ensures the exclusive-OR relation. Besides, DesignReviewSafety also consists of constraints for Roles, WorkProducts and Guidance. DesignReviewSafety does not have any further level.

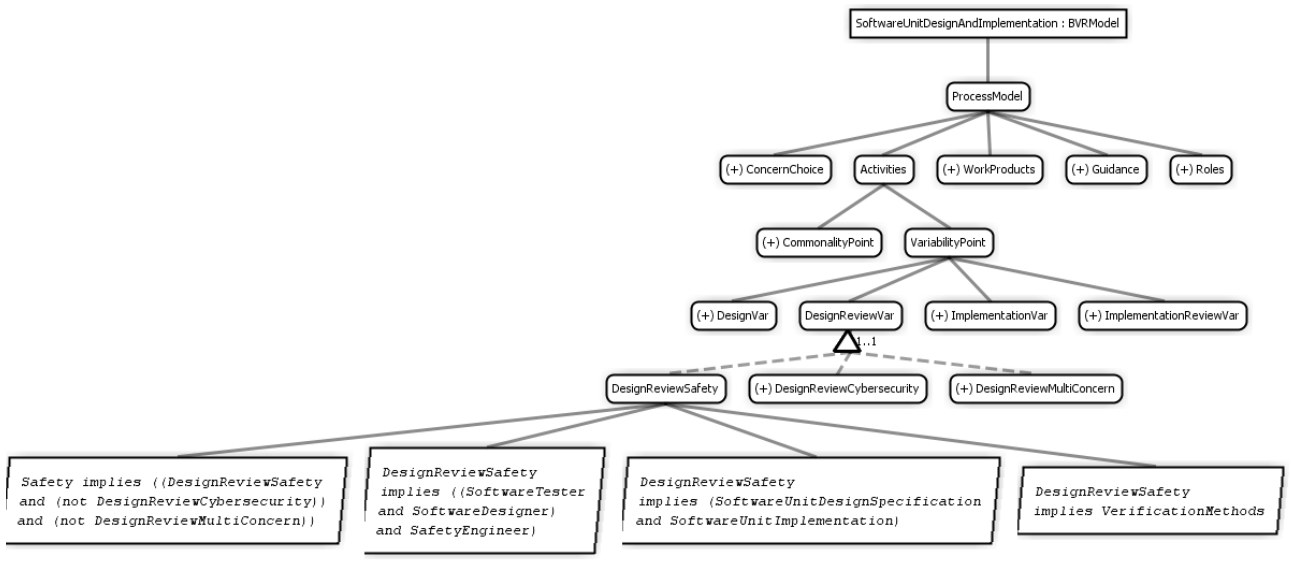

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
(+) DesignVar
DesignReviewVar
(+) ImplementationVar
(+) ImplementationReviewVar
1..1
DesignReviewSafety
(+) DesignReviewCybersecurity
(+) DesignReviewMultiConcern
Safety implies ((DesignReviewSafety and (not DesignReviewCybersecurity)) and (not DesignReviewMultiConcern))
DesignReviewSafety implies ((SoftwareTester and SoftwareDesigner) and SafetyEngineer)
DesignReviewSafety implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
DesignReviewSafety implies VerificationMethods

Figure 89. DesignReviewVar and DesignReviewSafety Subtrees

The DesignReviewVar and DesignReviewCybersecurity subtrees are depicted in Figure 90. The constraint restricting DesignReviewCybersecurity only to a choice of Cybersecurity for ConcernChoice ensures the exclusive-OR relation. Besides, DesignReviewCybersecurity also consists of constraints for Roles, WorkProducts and Guidance. DesignReviewCybersecurity does not have any further level.

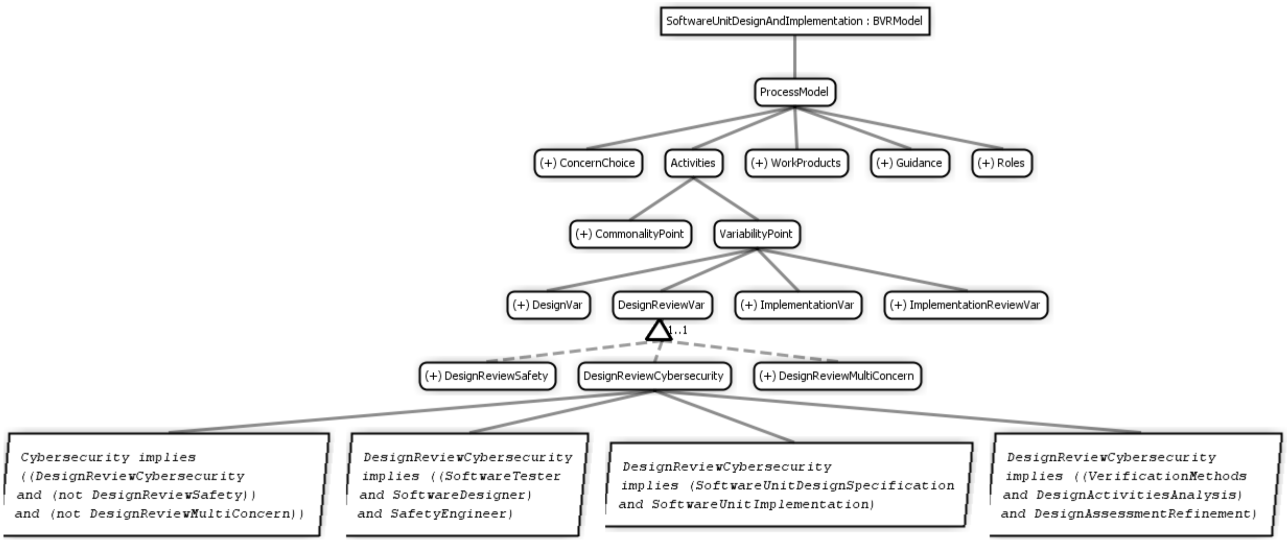

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
(+) DesignVar
DesignReviewVar
(+) ImplementationVar
(+) ImplementationReviewVar
1..1
(+) DesignReviewSafety
DesignReviewCybersecurity
(+) DesignReviewMultiConcern
Cybersecurity implies ((DesignReviewCybersecurity and (not DesignReviewSafety)) and (not DesignReviewMultiConcern))
DesignReviewCybersecurity implies ((SoftwareTester and SoftwareDesigner) and SafetyEngineer)
DesignReviewCybersecurity implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
DesignReviewCybersecurity implies ((VerificationMethods and DesignActivitiesAnalysis) and DesignAssessmentRefinement)

Figure 90. ReviewVar and DesignReviewCybersecurity Subtrees

The DesignReviewVar and DesignReviewMultiConcern subtrees are depicted in Figure 91. The constraint restricting DesignReviewMultiConcern only to a choice of MultiConcern for ConcernChoice ensures the exclusive-OR relation. Besides, DesignReviewMultiConcern also consists of constraints for Roles, WorkProducts and Guidance. DesignReviewMultiConcern also has a next level consisting of elements DesignReviewSafety and DesignReviewCybersecurity to cover multi concern design review tasks.

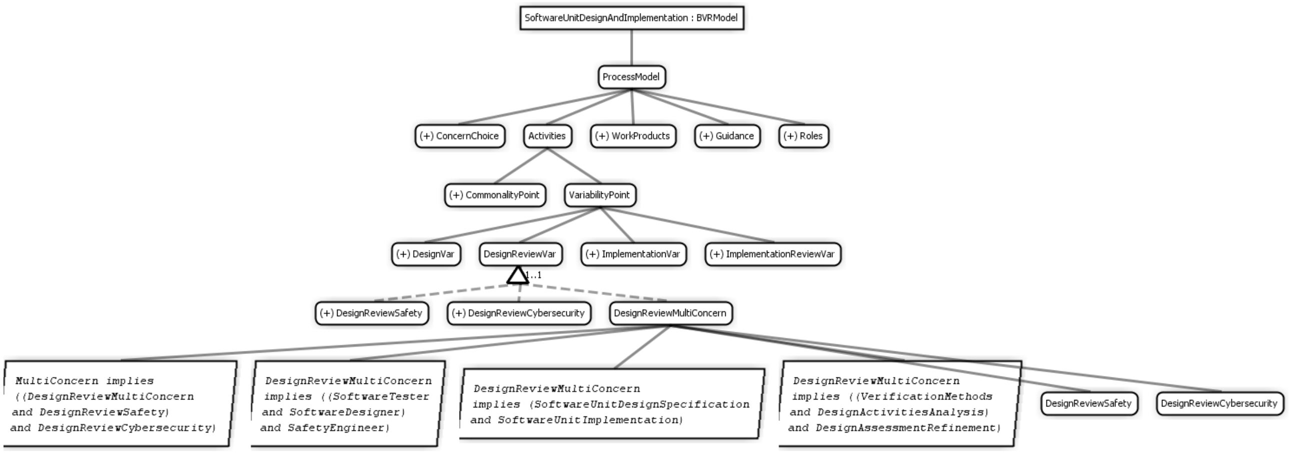

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
(+) DesignVar
DesignReviewVar
(+) ImplementationVar
(+) ImplementationReviewVar
1..1
(+) DesignReviewSafety
(+) DesignReviewCybersecurity
DesignReviewMultiConcern
MultiConcern implies ((DesignReviewMultiConcern and DesignReviewSafety) and DesignReviewCybersecurity)
DesignReviewMultiConcern implies ((SoftwareTester and SoftwareDesigner) and SafetyEngineer)
DesignReviewMultiConcern implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
DesignReviewMultiConcern implies ((VerificationMethods and DesignActivitiesAnalysis) and DesignAssessmentRefinement)
DesignReviewSafety
DesignReviewCybersecurity

Figure 91. DesignReviewVar and DesignReviewMultiConcern Subtrees

The ImplementationVar and ImplementationSafety subtrees are depicted in Figure 92. The ImplementationVar subtree consists of three elements, ImplementationSafety, ImplementationCybersecurity and ImplementationMultiConcern, related to each one of the choices of ConcernChoice and connected in exclusive-OR manner. The constraint restricting ImplementationSafety only to a choice of Safety for ConcernChoice ensures the exclusive-OR relation. Besides, ImplementationSafety also consists of constraints for Roles, WorkProducts and Guidance. ImplementationSafety does not have any further level.

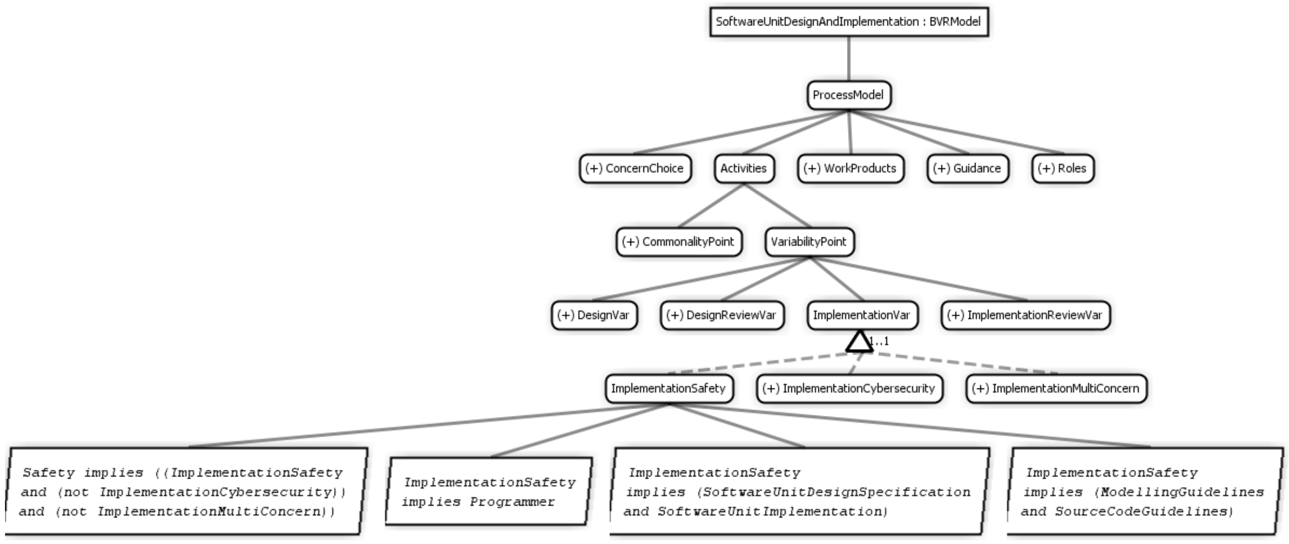

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
(+) DesignVar
(+) DesignReviewVar
ImplementationVar
(+) ImplementationReviewVar
1..1
ImplementationSafety
(+) ImplementationCybersecurity
(+) ImplementationMultiConcern
Safety implies ((ImplementationSafety and (not ImplementationCybersecurity)) and (not ImplementationMultiConcern))
ImplementationSafety implies Programmer
ImplementationSafety implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
ImplementationSafety implies (ModellingGuidelines and SourceCodeGuidelines)

Figure 92. ImplementationVar and ImplementationSafety Subtrees

The ImplementationVar and ImplementationCybersecurity subtrees are depicted in Figure 93. The constraint restricting ImplementationCybersecurity only to a choice of Cybersecurity for ConcernChoice ensures the exclusive-OR relation. Besides, ImplementationCybersecurity also consists of constraints for Roles, WorkProducts and Guidance. ImplementationCybersecurity does not have any further level.

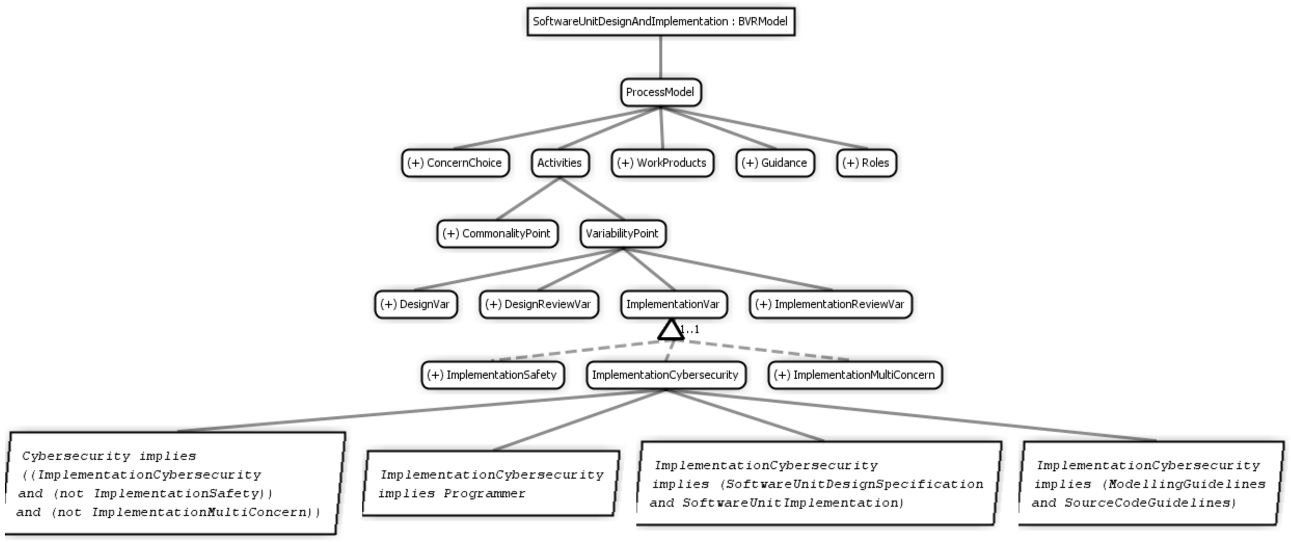

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
(+) DesignVar
(+) DesignReviewVar
ImplementationVar
(+) ImplementationReviewVar
1..1
(+) ImplementationSafety
ImplementationCybersecurity
(+) ImplementationMultiConcern
Cybersecurity implies ((ImplementationCybersecurity and (not ImplementationSafety)) and (not ImplementationMultiConcern))
ImplementationCybersecurity implies Programmer
ImplementationCybersecurity implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
ImplementationCybersecurity implies (ModellingGuidelines and SourceCodeGuidelines)

Figure 93. ImplementationVar and ImplementationCybersecurity Subtrees

The ImplementationVar and ImplementationMultiConcern subtrees are depicted in Figure 94. The constraint restricting ImplementationMultiConcern only to a choice of MultiConcern for ConcernChoice ensures the exclusive-OR relation. Besides, ImplementationMultiConcern also consists of constraints for Roles, WorkProducts and Guidance. ImplementationMultiConcern also has a next level consisting of elements ImplementationSafety and ImplementationCybersecurity to cover multi concern implementation tasks.

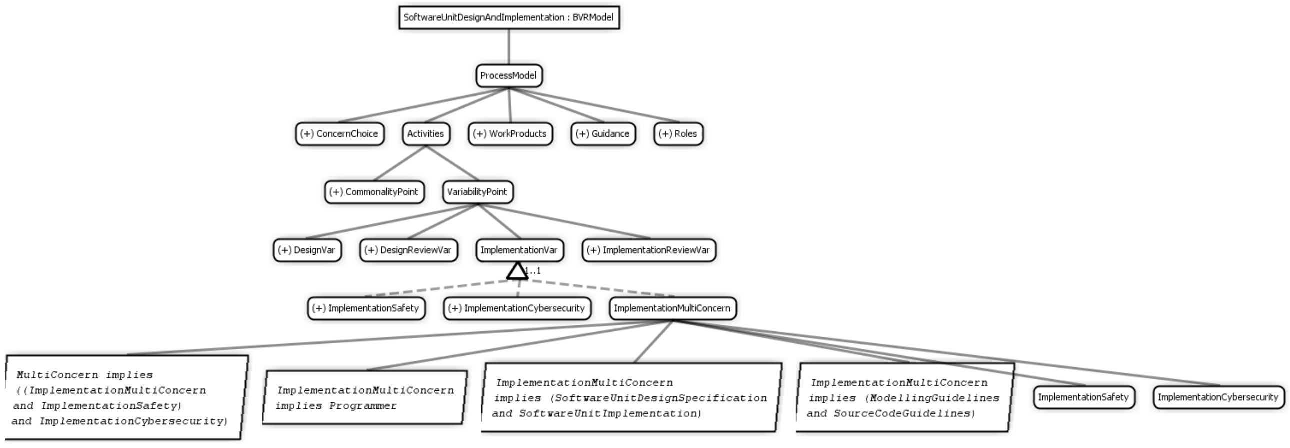

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
(+) DesignVar
(+) DesignReviewVar
ImplementationVar
(+) ImplementationReviewVar
1..1
(+) ImplementationSafety
(+) ImplementationCybersecurity
ImplementationMultiConcern
MultiConcern implies ((ImplementationMultiConcern and ImplementationSafety) and ImplementationCybersecurity)
ImplementationMultiConcern implies Programmer
ImplementationMultiConcern implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
ImplementationMultiConcern implies (ModellingGuidelines and SourceCodeGuidelines)
ImplementationSafety
ImplementationCybersecurity

Figure 94. ImplementationVar and ImplementationMultiConcern Subtrees

The ImplementationReviewVar and ImplementationReviewSafety subtrees are depicted in Figure 95. The ImplementationReviewVar subtree consists of three elements, ImplementationReviewSafety, ImplementationReviewCybersecurity and ImplementationReviewMultiConcern, related to each one of the choices of ConcernChoice and connected in exclusive-OR manner. The constraint restricting ImplementationReviewSafety only to a choice of Safety for ConcernChoice ensures the exclusive-OR relation. Besides, ImplementationReviewSafety also consists of constraints for Roles, WorkProducts and Guidance. ImplementationReviewSafety does not have any further level.

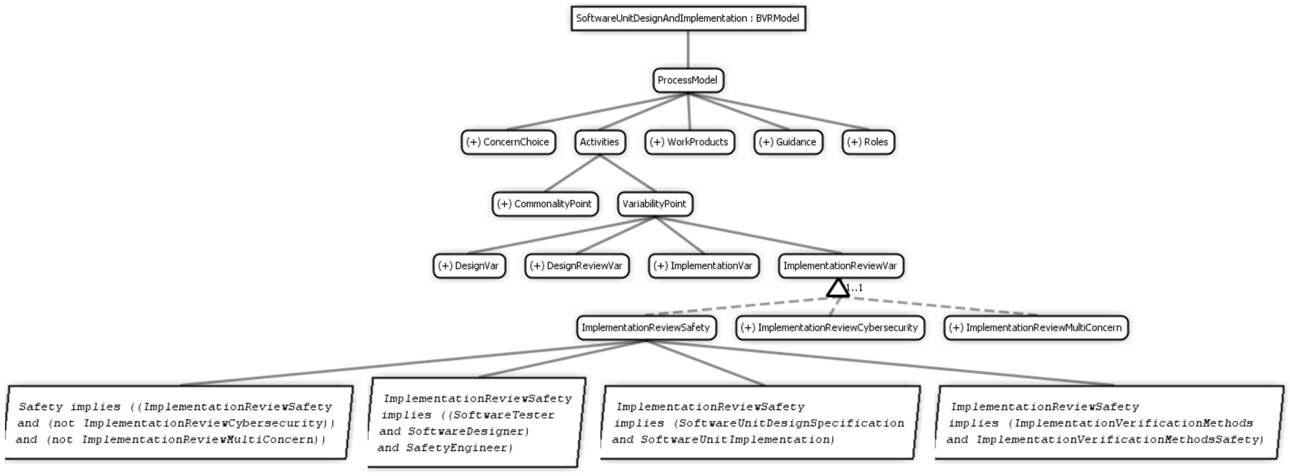
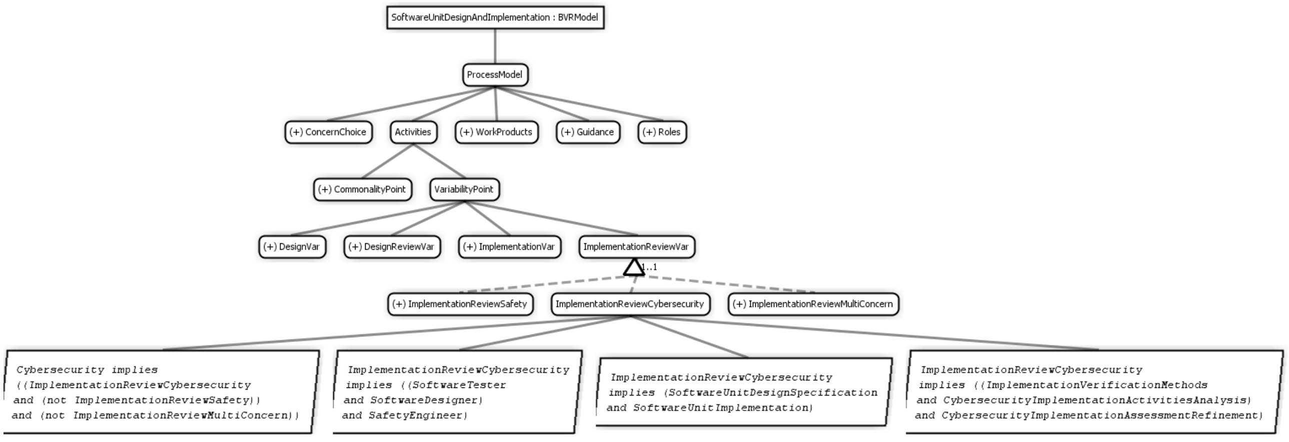

SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
(+) DesignVar
(+) DesignReviewVar
(+) ImplementationVar
ImplementationReviewVar
1..1
ImplementationReviewSafety
(+) ImplementationReviewCybersecurity
(+) ImplementationReviewMultiConcern
Safety implies ((ImplementationReviewSafety and (not ImplementationReviewCybersecurity)) and (not ImplementationReviewMultiConcern))
ImplementationReviewSafety implies ((SoftwareTester and SoftwareDesigner) and SafetyEngineer)
ImplementationReviewSafety implies (SoftwareUnitDesignSpecification and SoftwareUnitImplementation)
ImplementationReviewSafety implies (ImplementationVerificationMethods and ImplementationVerificationMethodsSafety)

Figure 95. ImplementationReviewVar and ImplementationReviewSafety Subtrees

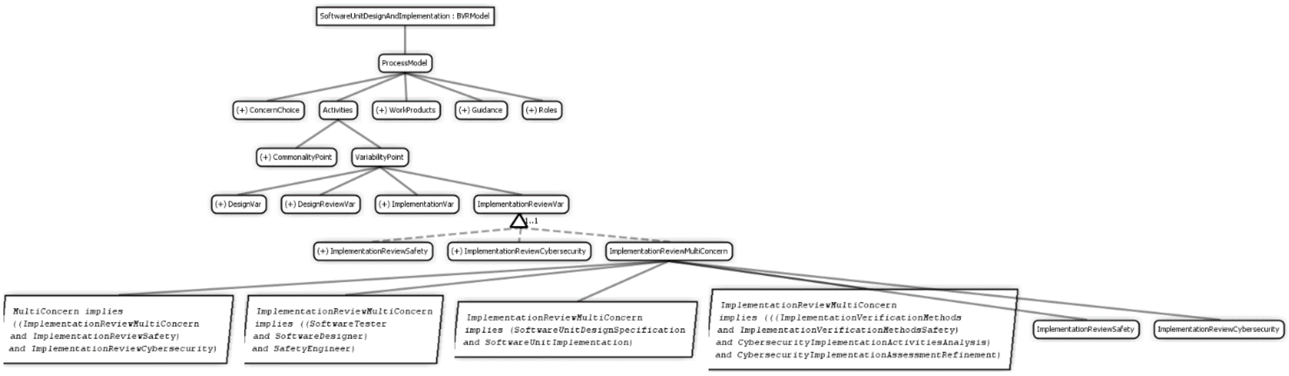
The ImplementationReviewVar and ImplementationReviewCybersecurity subtrees are depicted in Figure 96. The constraint restricting ImplementationReviewCybersecurity only to a choice of Cybersecurity for ConcernChoice ensures the exclusive-OR relation. Besides, ImplementationReviewCybersecurity also consists of constraints for Roles, WorkProducts and Guidance. ImplementationReviewCybersecurity does not have any further level.



```
Cybersecurity implies
((ImplementationReviewCybersecurity
and (not ImplementationReviewSafety))
and (not ImplementationReviewMultiConcern))

ImplementationReviewCybersecurity
implies ((SoftwareTester
and SoftwareDesigner)
and SafetyEngineer)

ImplementationReviewCybersecurity
implies (SoftwareUnitDesignSpecification
and SoftwareUnitImplementation)

ImplementationReviewCybersecurity
implies ((ImplementationVerificationMethods
and CybersecurityImplementationActivitiesAnalysis)
and CybersecurityImplementationAssessmentRefinement)
```

Figure 96. ImplementationReviewVar and ImplementationReviewCybersecurity Subtrees

The ImplementationReviewVar and ImplementationReviewMultiConcern subtrees are depicted in Figure 97. The constraint restricting ImplementationReviewMultiConcern only to a choice of MultiConcern for ConcernChoice ensures the exclusive-OR relation. Besides, ImplementationReviewMultiConcern also consists of constraints for Roles, WorkProducts and Guidance. ImplementationReviewMultiConcern also has a next level consisting of elements ImplementationReviewSafety and ImplementationReviewCybersecurity to cover multi concern implementation review tasks.


SoftwareUnitDesignAndImplementation : BVRModel
ProcessModel
(+) ConcernChoice
Activities
(+) WorkProducts
(+) Guidance
(+) Roles
(+) CommonalityPoint
VariabilityPoint
(+) DesignVar
(+) DesignReviewVar
(+) ImplementationVar
ImplementationReviewVar
1..1
(+) ImplementationReviewSafety
(+) ImplementationReviewCybersecurity
ImplementationReviewMultiConcern
ImplementationReviewSafety
ImplementationReviewCybersecurity

```
MultiConcern implies
((ImplementationReviewMultiConcern
and ImplementationReviewSafety)
and ImplementationReviewCybersecurity)

ImplementationReviewMultiConcern
implies ((SoftwareTester
and SoftwareDesigner)
and SafetyEngineer)

ImplementationReviewMultiConcern
implies (SoftwareUnitDesignSpecification
and SoftwareUnitImplementation)

ImplementationReviewMultiConcern
implies (((ImplementationVerificationMethods
and ImplementationVerificationMethodsSafety)
and CybersecurityImplementationActivitiesAnalysis)
and CybersecurityImplementationAssessmentRefinement)
```

Figure 97. ImplementationReviewVar and ImplementationReviewMultiConcern Subtrees

4.3.9. Configuration Resolution (Resolution Editor)

An example of a valid resolution of the Variability Model is depicted in Figure 98 for the choice 'MultiConcern'. All specified constraints are resolved correctly in the model depicted.

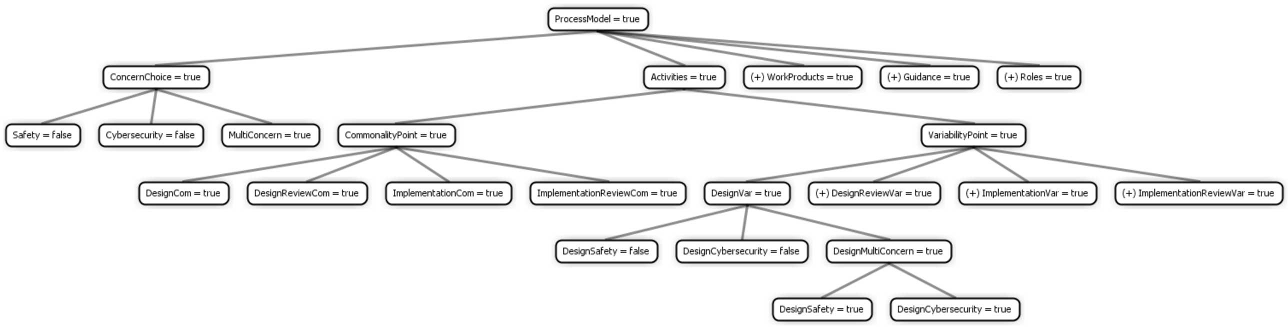


Figure 98. Example - Valid Resolution with ConcernChoice 'MultiConcern'

The same valid resolution showing cardinality is depicted in Figure 99.

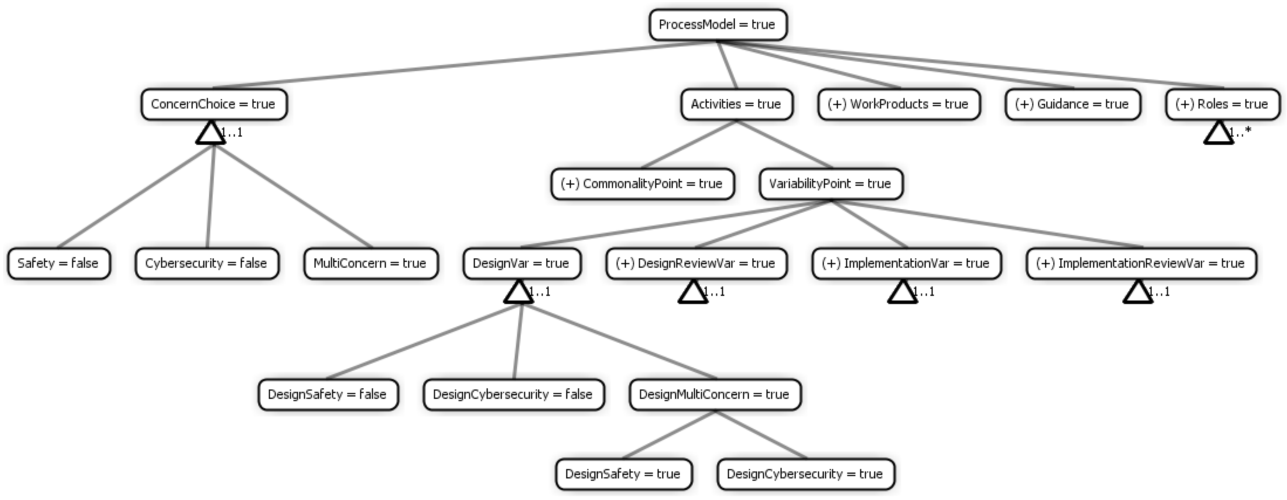


Figure 99. Example - Valid Resolution Showing Cardinality with ConcernChoice 'MultiConcern'

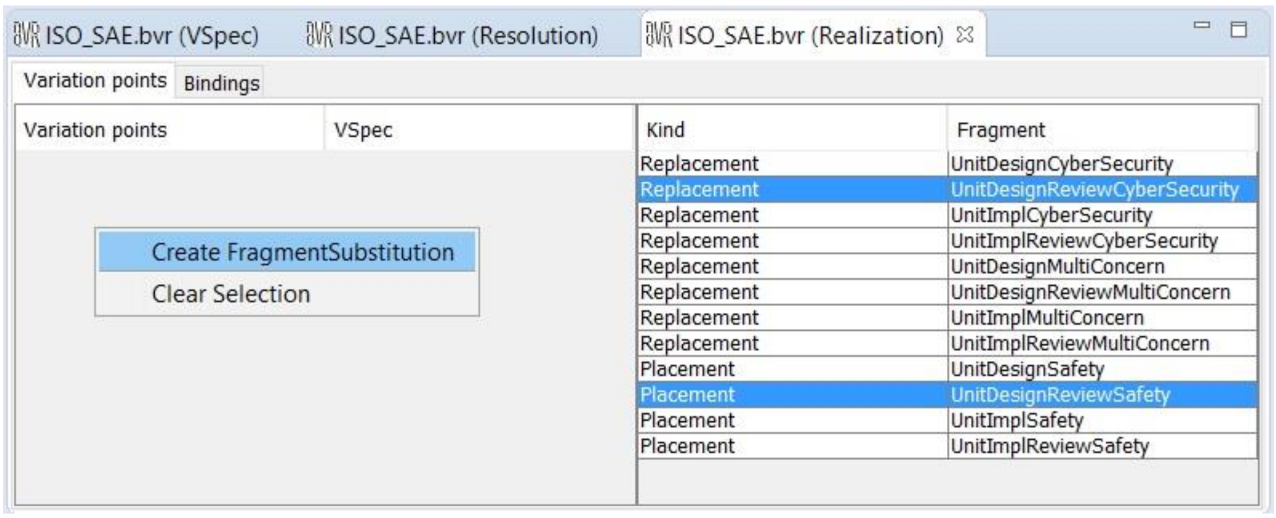
4.3.10. Model Realization (Realization Editor)

After the usage of the VSpec Editor (to build the Variability Model) and the Resolution Editor (to resolve valid configurations), variability management is continued by use of the Realization Editor.

The Base Model produced using the EPF Composer is used as input by the Realization Editor and fragment substitutions are applied to modify the Base Model according to the new resolved configuration.

For example, if the Base Model represents a safety-related (mono-concern = safety) process model (e.g., Figure 39) and if the mono-concern cybersecurity is chosen, the entire capability pattern for cybersecurity (shown in Figure 35) replaces (Replacement) the tasks concerning safety (Placement) in the base model (listed in Table 2).

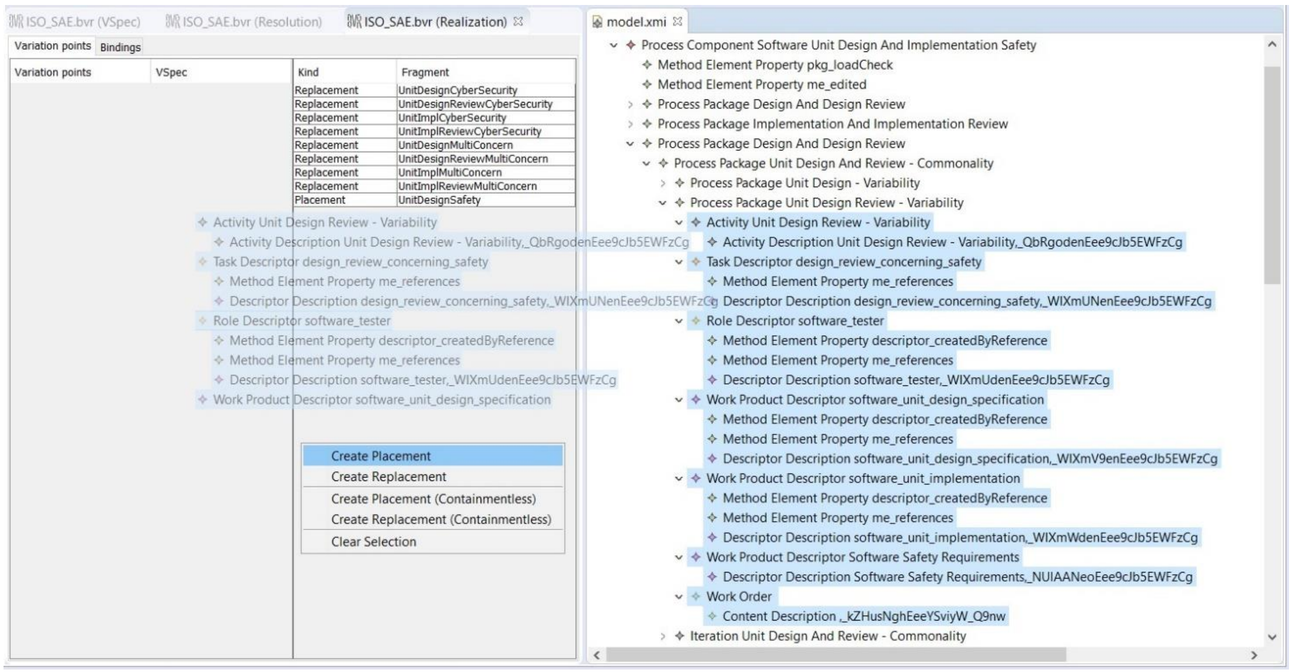
Figure 100 through Figure 104 illustrate the creation of fragment substitutions. Figure 100, for instance, illustrates the fragment substitution which indicates that the fragment UnitDesignReviewSafety (Placement) should be replaced by the fragment UnitDesignReviewCybersecurity (Replacement).



Kind	Fragment
Replacement	UnitDesignCyberSecurity
Replacement	UnitDesignReviewCyberSecurity
Replacement	UnitImplCyberSecurity
Replacement	UnitImplReviewCyberSecurity
Replacement	UnitDesignMultiConcern
Replacement	UnitDesignReviewMultiConcern
Replacement	UnitImplMultiConcern
Replacement	UnitImplReviewMultiConcern
Placement	UnitDesignSafety
Placement	UnitDesignReviewSafety
Placement	UnitImplSafety
Placement	UnitImplReviewSafety

Figure 100. Creation of Fragment Substitution

Figure 101 depicts the creation of Placements and Replacements. More specifically, in Figure 101, an example of the creation of a Placement (Unit Design Review) concerning Safety is shown. The associated activity, task descriptors, role descriptors and work product descriptors are displayed.

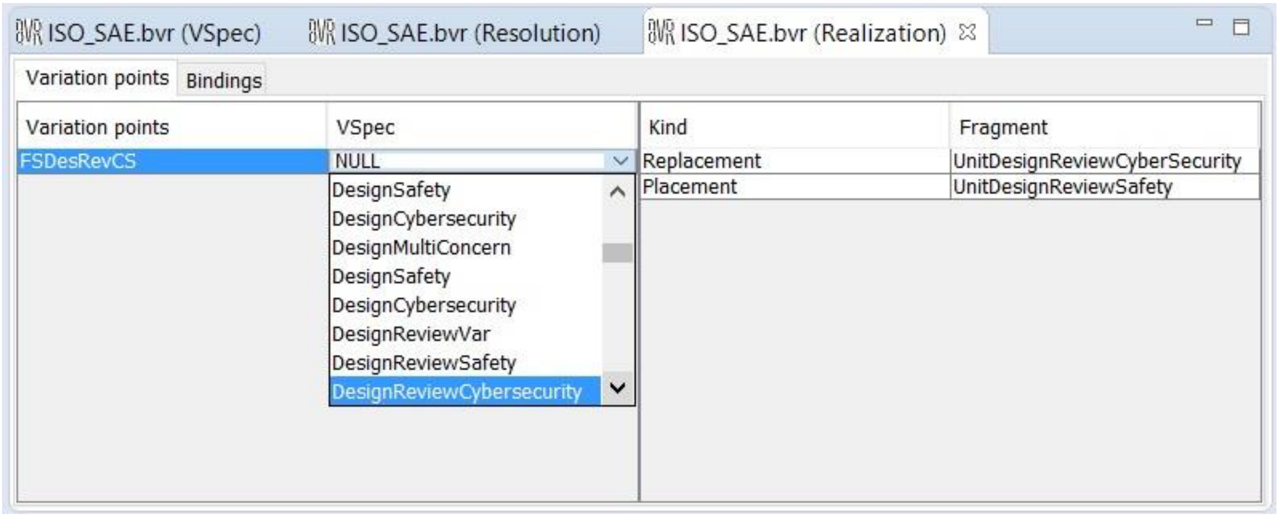


Variation points	VSpec	Kind	Fragment
		Replacement	UnitDesignCyberSecurity
		Replacement	UnitDesignReviewCyberSecurity
		Replacement	UnitImplCyberSecurity
		Replacement	UnitImplReviewCyberSecurity
		Replacement	UnitDesignMultiConcern
		Replacement	UnitDesignReviewMultiConcern
		Replacement	UnitImplMultiConcern
		Replacement	UnitImplReviewMultiConcern
		Placement	UnitDesignSafety

```
Process Component Software Unit Design And Implementation Safety
  Method Element Property pkg_loadCheck
  Method Element Property me_edited
  Process Package Design And Design Review
  Process Package Implementation And Implementation Review
  Process Package Design And Design Review
    Process Package Unit Design And Review - Commonality
      Process Package Unit Design - Variability
      Process Package Unit Design Review - Variability
        Activity Unit Design Review - Variability
          Activity Description Unit Design Review - Variability,_QbRgodenEee9cJb5EWFzCg
        Task Descriptor design_review_concerning_safety
          Method Element Property me_references
          Descriptor Description design_review_concerning_safety,_WIXmUNenEee9cJb5EWFzCg
        Role Descriptor software_tester
          Method Element Property descriptor_createdByReference
          Method Element Property me_references
          Descriptor Description software_tester,_WIXmUdenEee9cJb5EWFzCg
        Work Product Descriptor software_unit_design_specification
          Method Element Property descriptor_createdByReference
          Method Element Property me_references
          Descriptor Description software_unit_design_specification,_WIXmV9enEee9cJb5EWFzCg
        Work Product Descriptor software_unit_implementation
          Method Element Property descriptor_createdByReference
          Method Element Property me_references
          Descriptor Description software_unit_implementation,_WIXmWdenEee9cJb5EWFzCg
        Work Product Descriptor Software Safety Requirements
          Descriptor Description Software Safety Requirements,_NUIAANeoEee9cJb5EWFzCg
        Work Order
          Content Description ,_kZHusNghEeeYSviyW_Q9nw
      Iteration Unit Design And Review - Commonality
```

Figure 101. Creation of Placement/Replacement

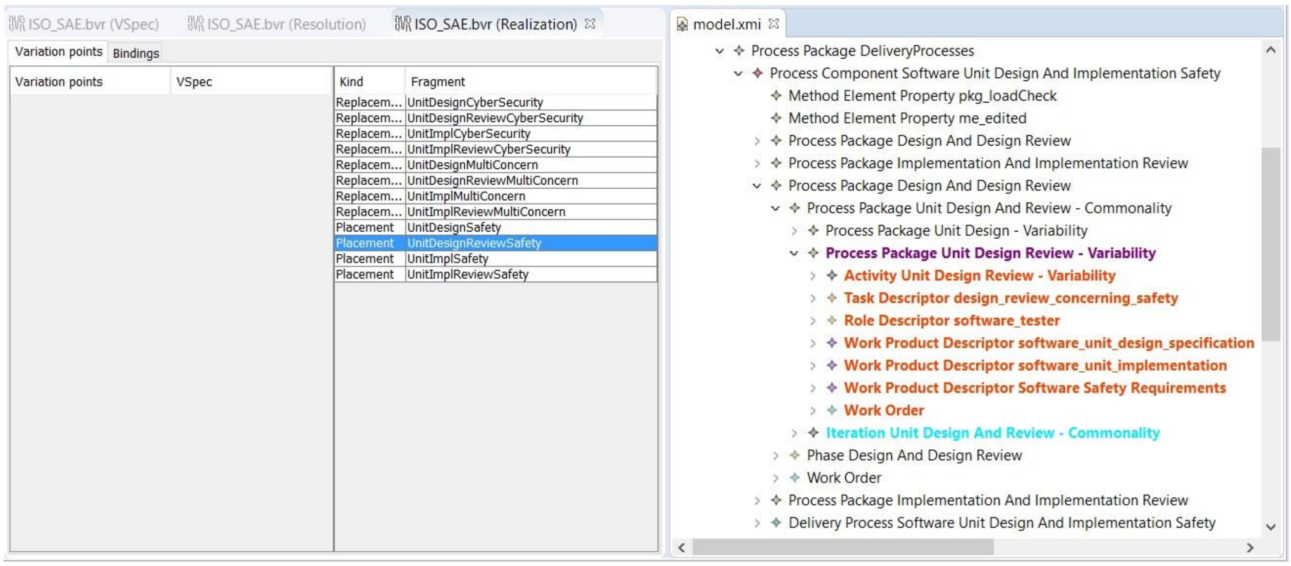
Figure 102 depicts the binding between the elements (abstract features) created using the VSpec editor with the chosen concrete fragments. The elements created in the VSpec editor correspond to the elements depicted in Figure 90 for Design Review concerning Cybersecurity.



Variation points	VSpec	Kind	Fragment
FSDesRevCS	NULL	Replacement	UnitDesignReviewCyberSecurity
		Placement	UnitDesignReviewSafety

Figure 102. Linking VSpec to Fragment Substitution

In this example, Unit Design Review Safety is the substitution fragment which is selected as a Placement fragment, and hence, is to be removed from the model being realized. This is depicted in Figure 103.


ISO_SAE.bvr (VSpec) ISO_SAE.bvr (Resolution) ISO_SAE.bvr (Realization)

Variation points | Bindings

Variation points	VSpec

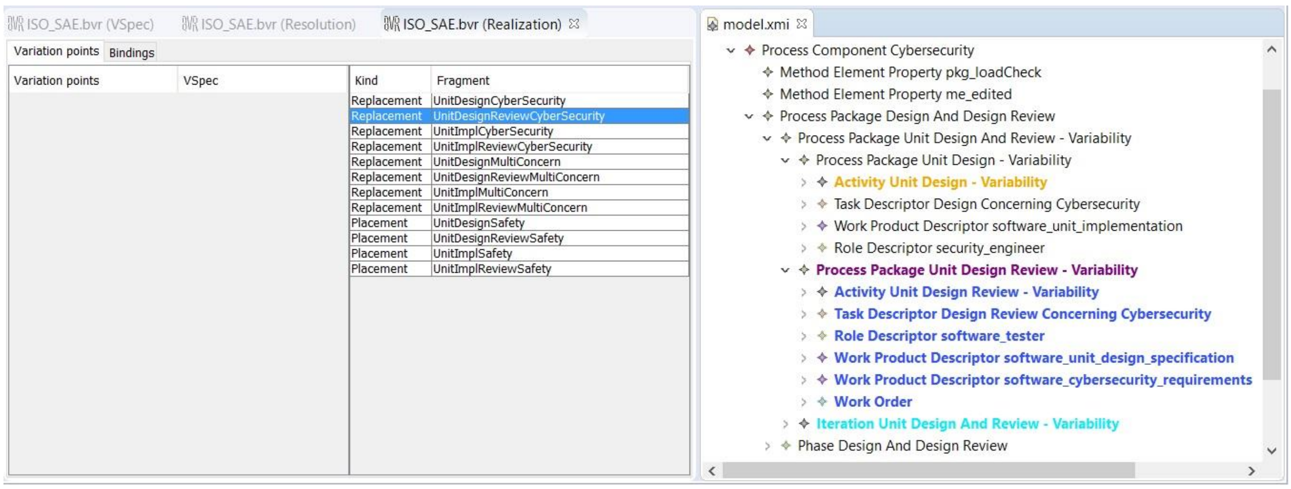
Kind	Fragment
Replacem...	UnitDesignCyberSecurity
Replacem...	UnitDesignReviewCyberSecurity
Replacem...	UnitImplCyberSecurity
Replacem...	UnitImplReviewCyberSecurity
Replacem...	UnitDesignMultiConcern
Replacem...	UnitDesignReviewMultiConcern
Replacem...	UnitImplMultiConcern
Replacem...	UnitImplReviewMultiConcern
Placement	UnitDesignSafety
Placement	UnitDesignReviewSafety
Placement	UnitImplSafety
Placement	UnitImplReviewSafety

model.xmi

```
Process Package DeliveryProcesses
  Process Component Software Unit Design And Implementation Safety
    Method Element Property pkg_loadCheck
    Method Element Property me_edited
    Process Package Design And Design Review
    Process Package Implementation And Implementation Review
    Process Package Design And Design Review
      Process Package Unit Design And Review - Commonality
        Process Package Unit Design - Variability
        Process Package Unit Design Review - Variability
          Activity Unit Design Review - Variability
          Task Descriptor design_review_concerning_safety
          Role Descriptor software_tester
          Work Product Descriptor software_unit_design_specification
          Work Product Descriptor software_unit_implementation
          Work Product Descriptor Software Safety Requirements
          Work Order
        Iteration Unit Design And Review - Commonality
      Phase Design And Design Review
      Work Order
    Process Package Implementation And Implementation Review
    Delivery Process Software Unit Design And Implementation Safety
```

Figure 103. Placement Unit Design Review Safety

Also in this example, Unit Design Review Cybersecurity is the substitution fragment which is selected as a Replacement fragment, and hence is to be added to the model being realized. This is depicted in Figure 104.


ISO_SAE.bvr (VSpec) ISO_SAE.bvr (Resolution) ISO_SAE.bvr (Realization)

Variation points | Bindings

Variation points	VSpec

Kind	Fragment
Replacement	UnitDesignCyberSecurity
Replacement	UnitDesignReviewCyberSecurity
Replacement	UnitImplCyberSecurity
Replacement	UnitImplReviewCyberSecurity
Replacement	UnitDesignMultiConcern
Replacement	UnitDesignReviewMultiConcern
Replacement	UnitImplMultiConcern
Replacement	UnitImplReviewMultiConcern
Placement	UnitDesignSafety
Placement	UnitDesignReviewSafety
Placement	UnitImplSafety
Placement	UnitImplReviewSafety

model.xmi

```
Process Component Cybersecurity
  Method Element Property pkg_loadCheck
  Method Element Property me_edited
  Process Package Design And Design Review
    Process Package Unit Design And Review - Variability
      Process Package Unit Design - Variability
        Activity Unit Design - Variability
        Task Descriptor Design Concerning Cybersecurity
        Work Product Descriptor software_unit_implementation
        Role Descriptor security_engineer
      Process Package Unit Design Review - Variability
        Activity Unit Design Review - Variability
        Task Descriptor Design Review Concerning Cybersecurity
        Role Descriptor software_tester
        Work Product Descriptor software_unit_design_specification
        Work Product Descriptor software_cybersecurity_requirements
        Work Order
      Iteration Unit Design And Review - Variability
    Phase Design And Design Review
```

Figure 104. Replacement Unit Design Review Cybersecurity

The Realization Model created using the Realization Editor by a series of substitutions consisting of Placements and Replacements can be exported back to the Base Model Editor (EPF Composer).

The consolidated activity diagram of the realized cybersecurity model is depicted in Figure 105 through Figure 107. The activity diagram is split into three parts to enhance readability. Figure 105 depicts the phases, iterations and activities of the delivery process. The phases are executed sequentially. The activities which make up each phase are executed iteratively as described. The tasks which make up the activities are depicted in Figure 106 and Figure 107. Figure 106 depicts the Commonality tasks while Figure 107 depicts the Variability tasks related to Cybersecurity.

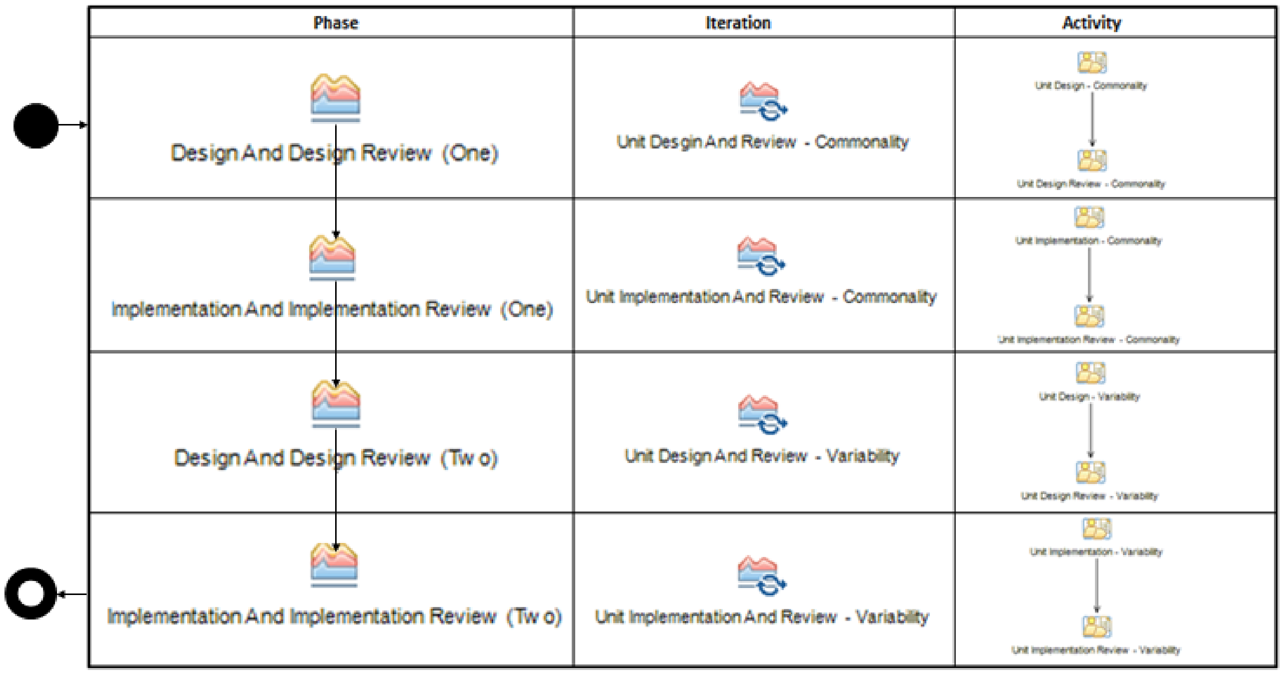

Phase
Iteration
Activity
Design And Design Review (One)
Unit Desgin And Review - Commonality
Unit Design - Commonality
Unit Design Review - Commonality
Implementation And Implementation Review (One)
Unit Implementation And Review - Commonality
Unit Implementation - Commonality
Unit Implementation Review - Commonality
Design And Design Review (Two)
Unit Design And Review - Variability
Unit Design - Variability
Unit Design Review - Variability
Implementation And Implementation Review (Two)
Unit Implementation And Review - Variability
Unit Implementation - Variability
Unit Implementation Review - Variability

Figure 105. Detailed Activity Diagram Cybersecurity (1 of 3)

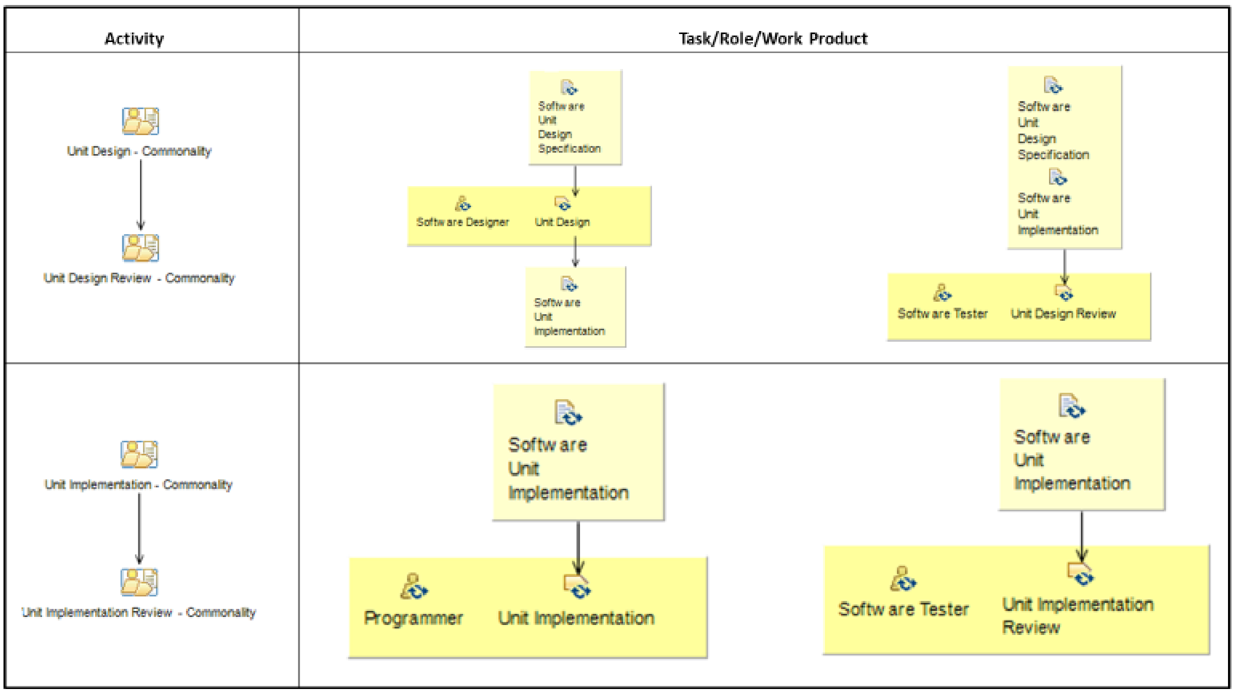

Activity
Task/Role/Work Product
Unit Design - Commonality
Unit Design Review - Commonality
Software Unit Design Specification
Software Designer
Unit Design
Software Unit Implementation
Software Unit Design Specification
Software Unit Implementation
Software Tester
Unit Design Review
Unit Implementation - Commonality
Unit Implementation Review - Commonality
Software Unit Implementation
Programmer
Unit Implementation
Software Unit Implementation
Software Tester
Unit Implementation Review

Figure 106. Detailed Activity Diagram Cybersecurity (2 of 3)

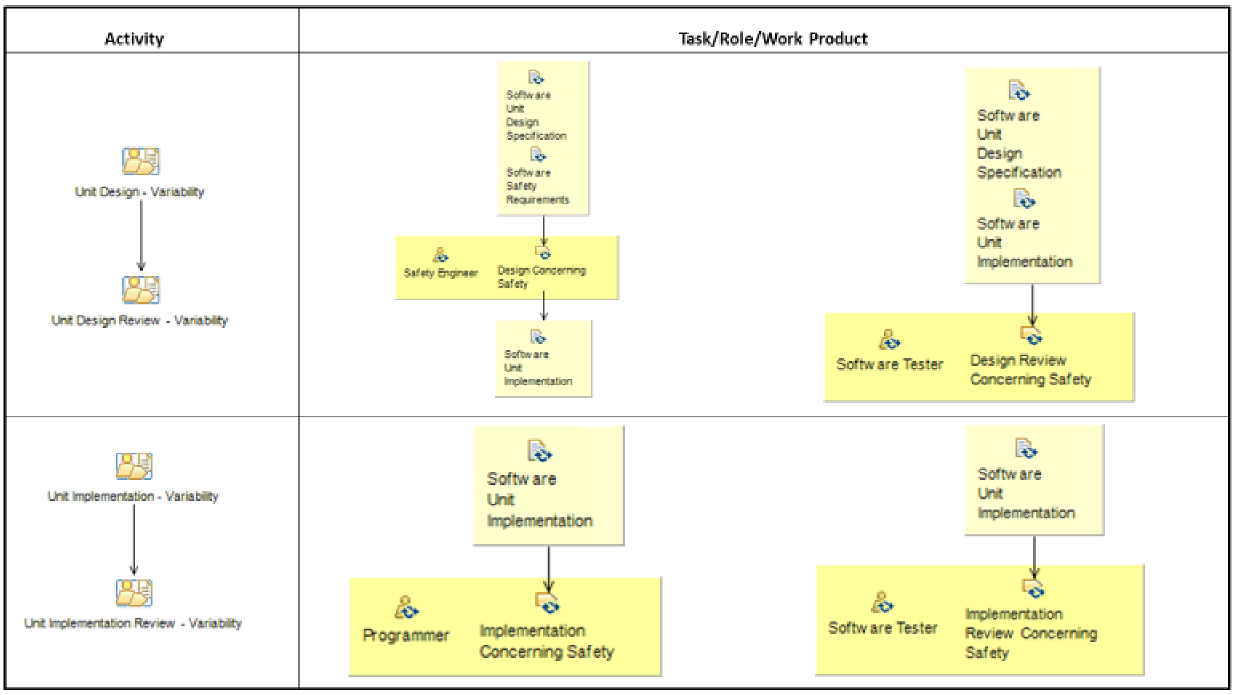

Activity
Task/Role/Work Product
Unit Design - Variability
Unit Design Review - Variability
Software Unit Design Specification
Software Safety Requirements
Safety Engineer
Design Concerning Safety
Software Unit Implementation
Software Tester
Design Review Concerning Safety
Unit Implementation - Variability
Unit Implementation Review - Variability
Programmer
Implementation Concerning Safety
Implementation Review Concerning Safety

Figure 107. Detailed Activity Diagram Cybersecurity (3 of 3)

The consolidated activity diagram of the realized multi concern model is depicted in Figure 108 through Figure 110. The activity diagram is split into three parts to enhance readability. Only the Variability elements for multi concern are depicted as the elements shown in Figure 105 and Figure 106 are the same for multi concern. The diagram corresponds to our interpretation of the 'Pattern Engineering Lifecycle' depicted in Figure 37**.**

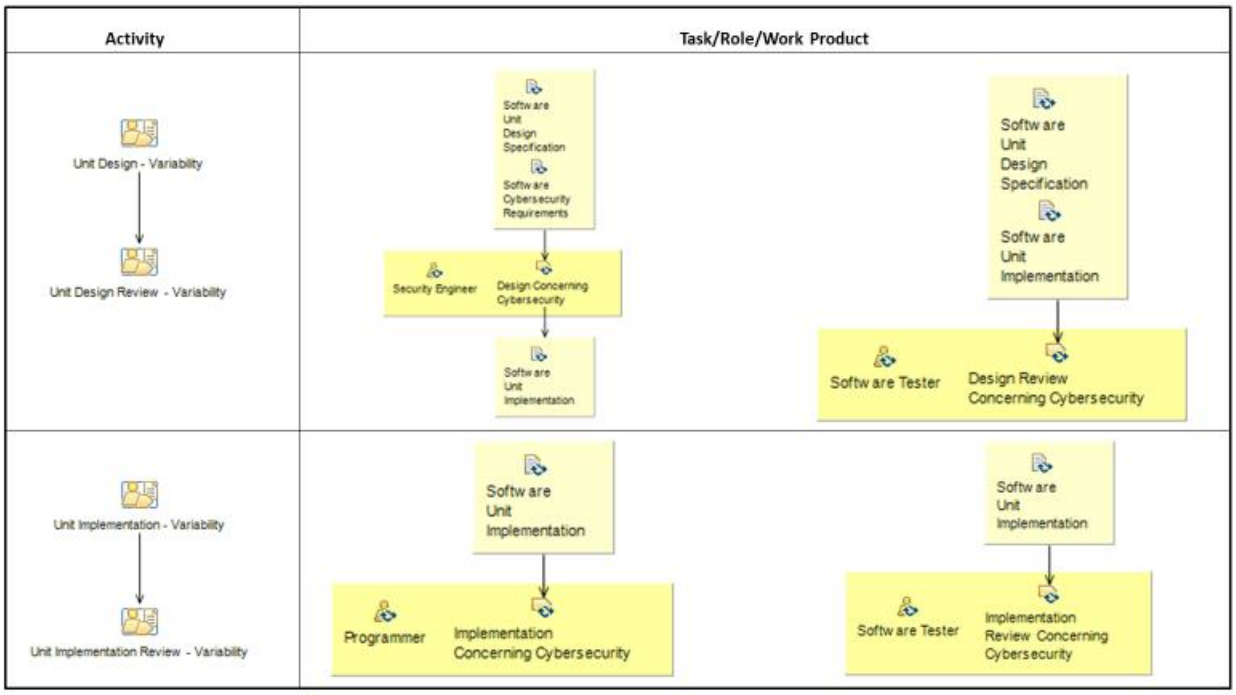

Activity
Task/Role/Work Product
Unit Design - Variability
Unit Design Review - Variability
Software Unit Design Specification
Software Cybersecurity Requirements
Security Engineer
Design Concerning Cybersecurity
Software Unit Implementation
Software Tester
Design Review Concerning Cybersecurity
Unit Implementation - Variability
Unit Implementation Review - Variability
Programmer
Implementation Concerning Cybersecurity
Implementation Review Concerning Cybersecurity

Figure 108. Detailed Activity Diagram Multi Concern (1 of 3)

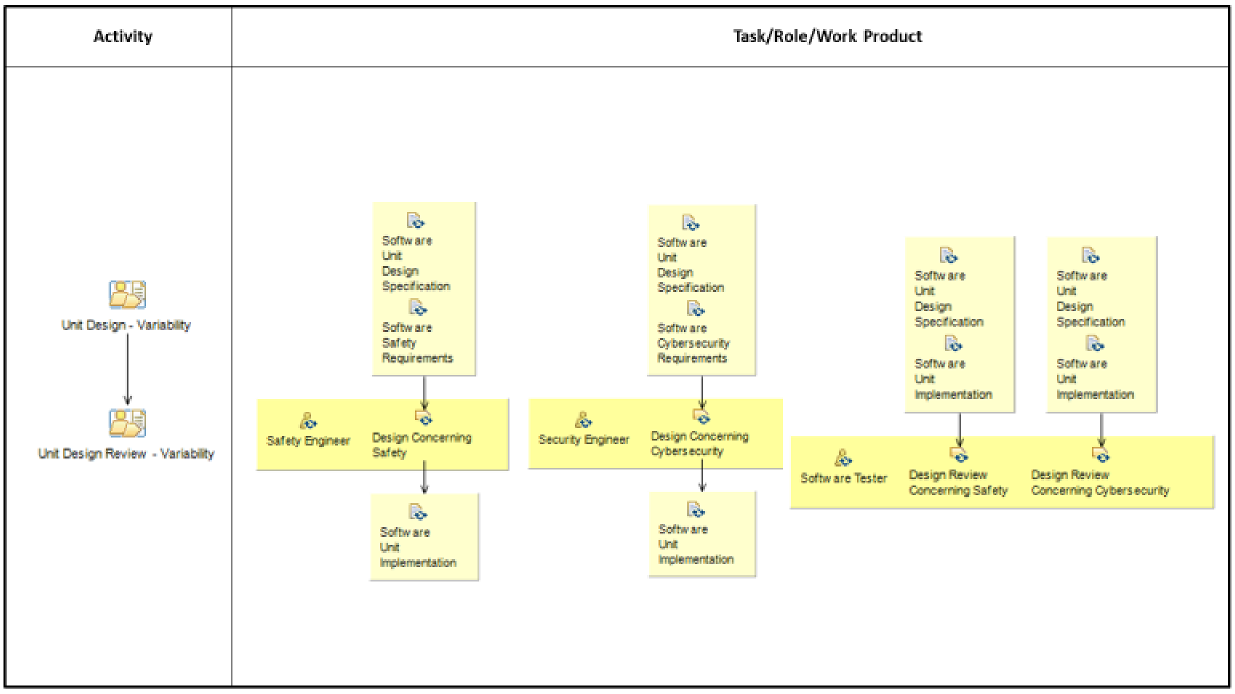

Activity
Task/Role/Work Product
Unit Design - Variability
Unit Design Review - Variability
Software Unit Design Specification
Software Safety Requirements
Safety Engineer
Design Concerning Safety
Software Unit Implementation
Software Cybersecurity Requirements
Security Engineer
Design Concerning Cybersecurity
Software Tester
Design Review Concerning Safety
Design Review Concerning Cybersecurity

Figure 109. Detailed Activity Diagram Multi Concern (2 of 3)

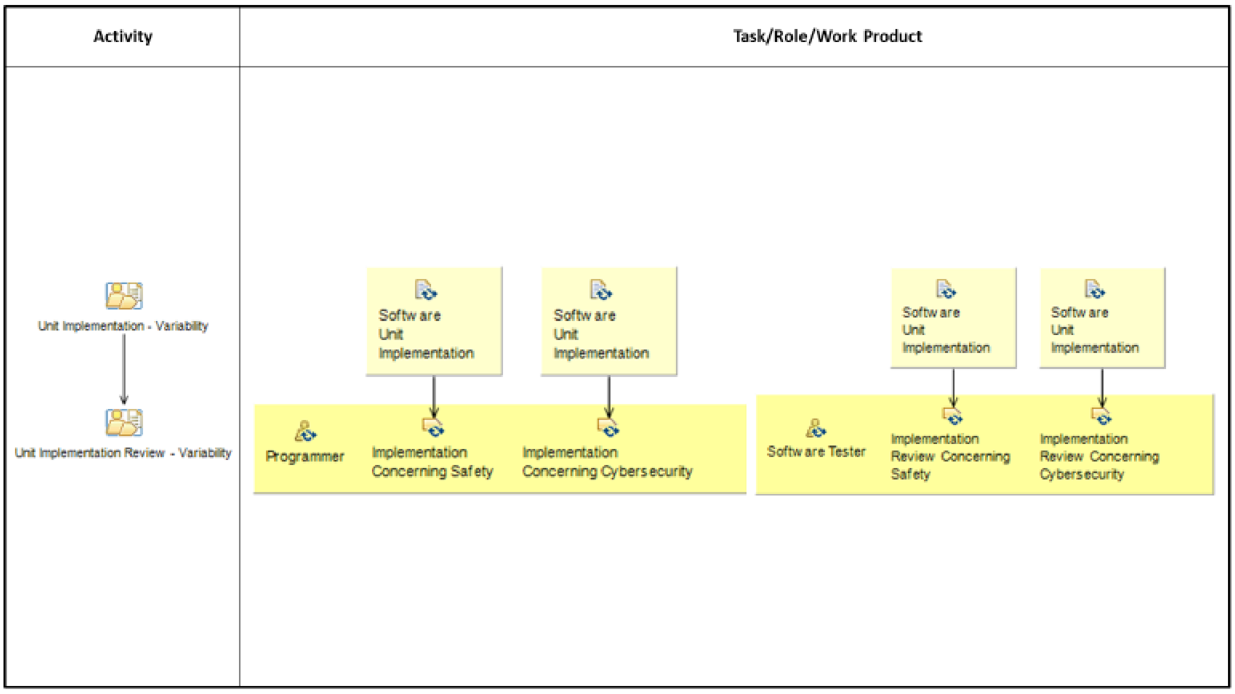

Activity
Task/Role/Work Product
Unit Implementation - Variability
Unit Implementation Review - Variability
Software Unit Implementation
Programmer
Implementation Concerning Safety
Implementation Concerning Cybersecurity
Software Tester
Implementation Review Concerning Safety
Implementation Review Concerning Cybersecurity

Figure 110. Detailed Activity Diagram Multi Concern (3 of 3)

4.3.11. Case Study Conclusion

The parts of the use case modelled in the EPF Composer provide a process model and the related process-related assurance assets for co-assessment. The parts of the use case modelled in the BVR tool provide a means to deal with cross-concern reuse/co-assessment of the software development process factoring safety and cybersecurity requirements.

4.4. CS1: Industrial and Automation Control Systems (IACS) (*)

In this section, the CS1: Industrial and Automation Control Systems (IACS) described in D1.1 [1] is considered to illustrate the dependability co-analysis via Safety Architect. Two use case scenarios have been defined for the CS1: US1 (Compliance Management), US2 (Safety and security co-assessment). The illustration of the dependability co-analysis via Safety Architect is explained for US2.

4.4.1. Description of the Use Case Scenario

We have considered Schneider Electric Saitel® RTUs – high-level architecture, depicted in Figure 111, to demonstrate the system safety and security co-analysis methodology presented in Section 3.5.2.

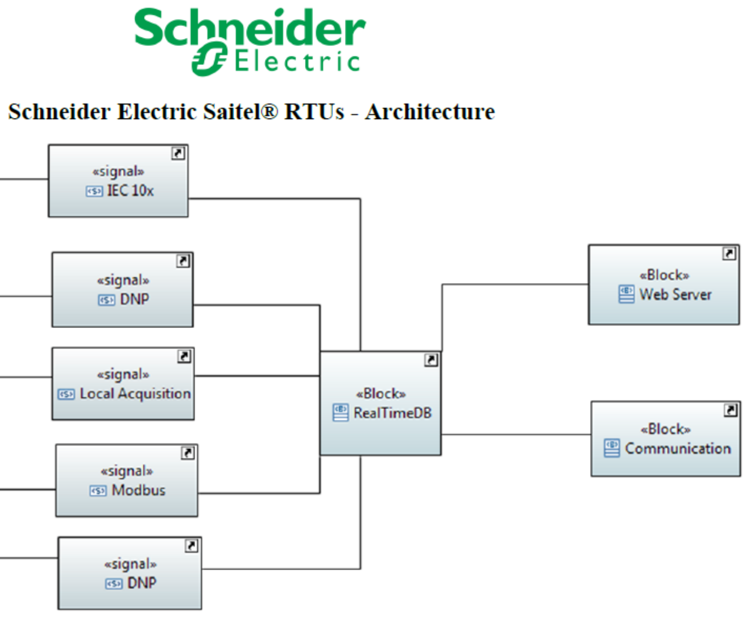


Figure 111. High-level Archiecture of SchneiderElectric Saitel RTU

The objective of CS1-US2 (Safety and security co-assessment) is to support the RTU Safety & Security Assurance Case with AMASS platform and dedicated external tools. In this use case, the Safety Architect and Cyber Architect tools are integrated with the AMASS platform as external tools to provide this safety and security co-analysis support.

4.4.2. Demonstration of the Methodology

As explained in Section 3.5.2 (System Dependability Co-analysis via Safety Architect), a possible usage scenario for Safety & Security co-analysis is composed by 8 steps. These steps are illustrated below:

Step 1: System engineers can design its system architecture model with CHESS tool. The Schneider Electric Saitel® RTUs – high-level architecture model in CHESS is shown in Figure 112 .

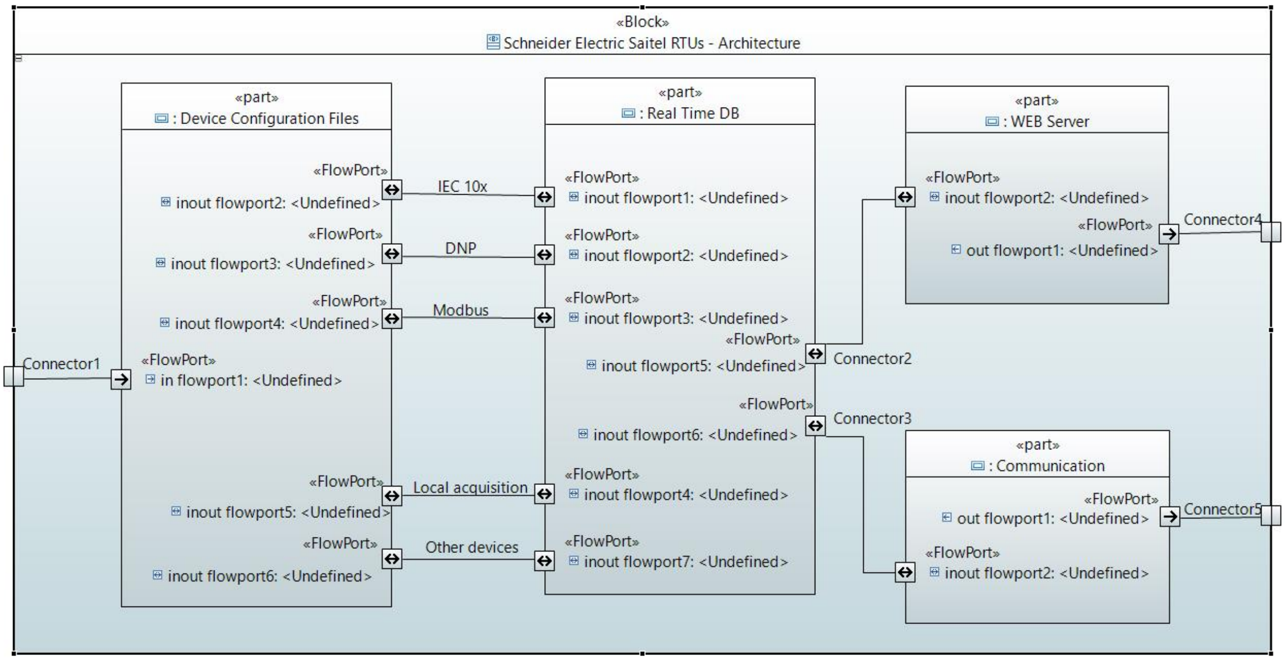
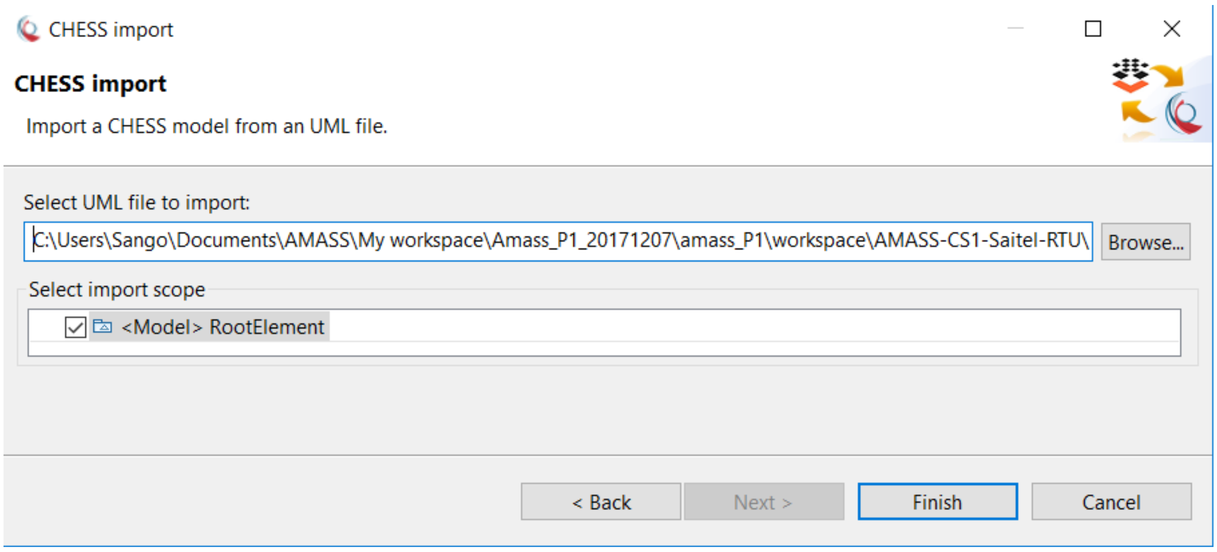

«Block»
Schneider Electric Saitel RTUs - Architecture
«part»
: Device Configuration Files
«FlowPort» inout flowport2: <Undefined>
«FlowPort» inout flowport3: <Undefined>
«FlowPort» inout flowport4: <Undefined>
«FlowPort» in flowport1: <Undefined>
«FlowPort» inout flowport5: <Undefined>
«FlowPort» inout flowport6: <Undefined>
Connector1
IEC 10x
DNP
Modbus
Local acquisition
Other devices
«part»
: Real Time DB
«FlowPort» inout flowport1: <Undefined>
«FlowPort» inout flowport2: <Undefined>
«FlowPort» inout flowport3: <Undefined>
«FlowPort» inout flowport5: <Undefined>
«FlowPort» inout flowport6: <Undefined>
«FlowPort» inout flowport4: <Undefined>
«FlowPort» inout flowport7: <Undefined>
Connector2
Connector3
«part»
: WEB Server
«FlowPort» inout flowport2: <Undefined>
«FlowPort» out flowport1: <Undefined>
Connector4
«part»
: Communication
«FlowPort» out flowport1: <Undefined>
«FlowPort» inout flowport2: <Undefined>
Connector5

Figure 112. SchneiderElectric Saitel RTU High-level Architecture model in CHESS

Step 2: Safety Engineers can import system model from the AMASS platform – CHESS tool to Safety Architect tool for safety analysis thanks to the connector between CHESS and Safety Architect, as illustrated in Figure 113 and Figure 114.


CHESS import

CHESS import

Import a CHESS model from an UML file.

Select UML file to import:

C:\Users\Sango\Documents\AMASS\My workspace\Amass_P1_20171207\amass_P1\workspace\AMASS-CS1-Saitel-RTU\

Select import scope

☑ <Model> RootElement

Figure 113. Import from CHESS tool to Safety Architect tool

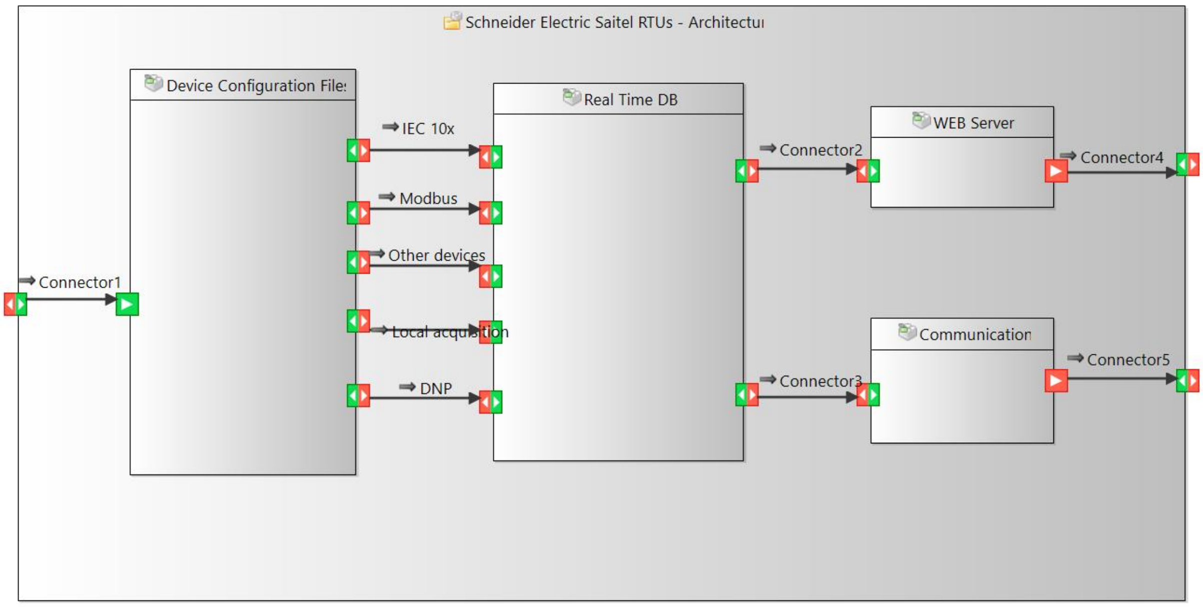
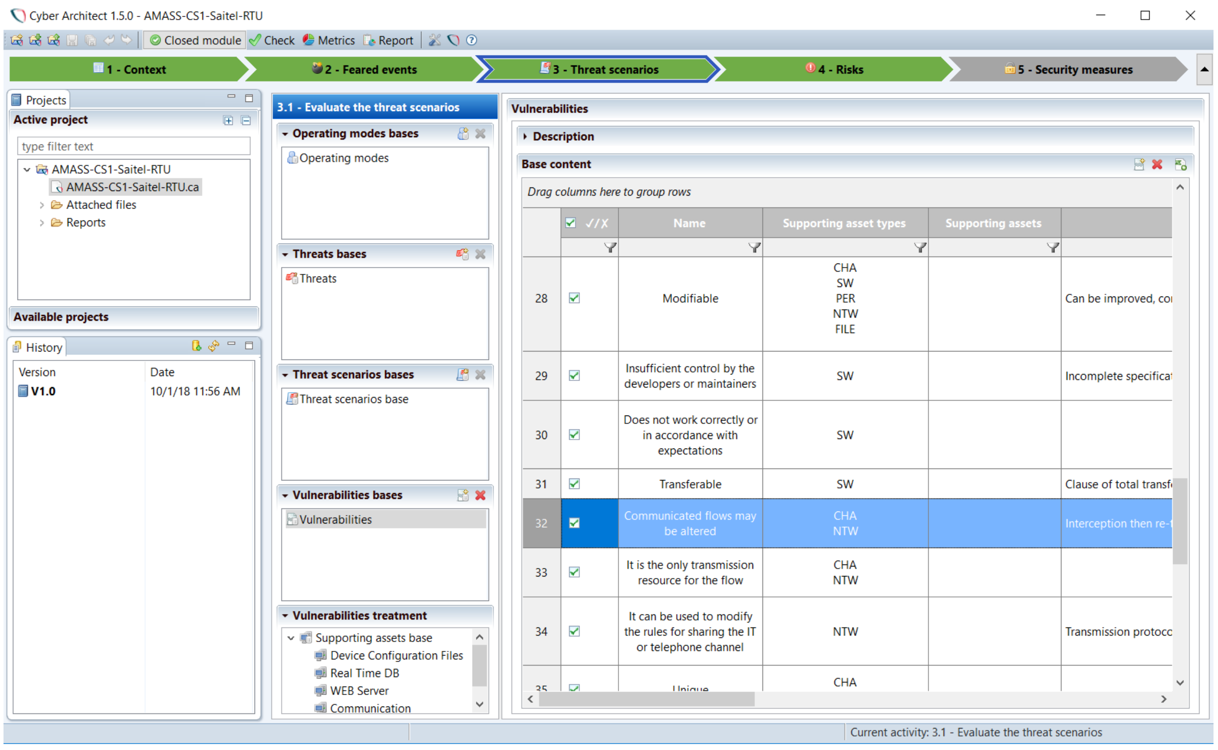


Figure 114. Safety Architect WBS model from CHESS WBS model

Step 3: Security engineers (in parallel to safety analysis or based on the process defined in other AMASS platform tools, such as EPF or external tools, such as WEFACT) can perform its security analysis in Safety Architect as illustrated in Figure 115.

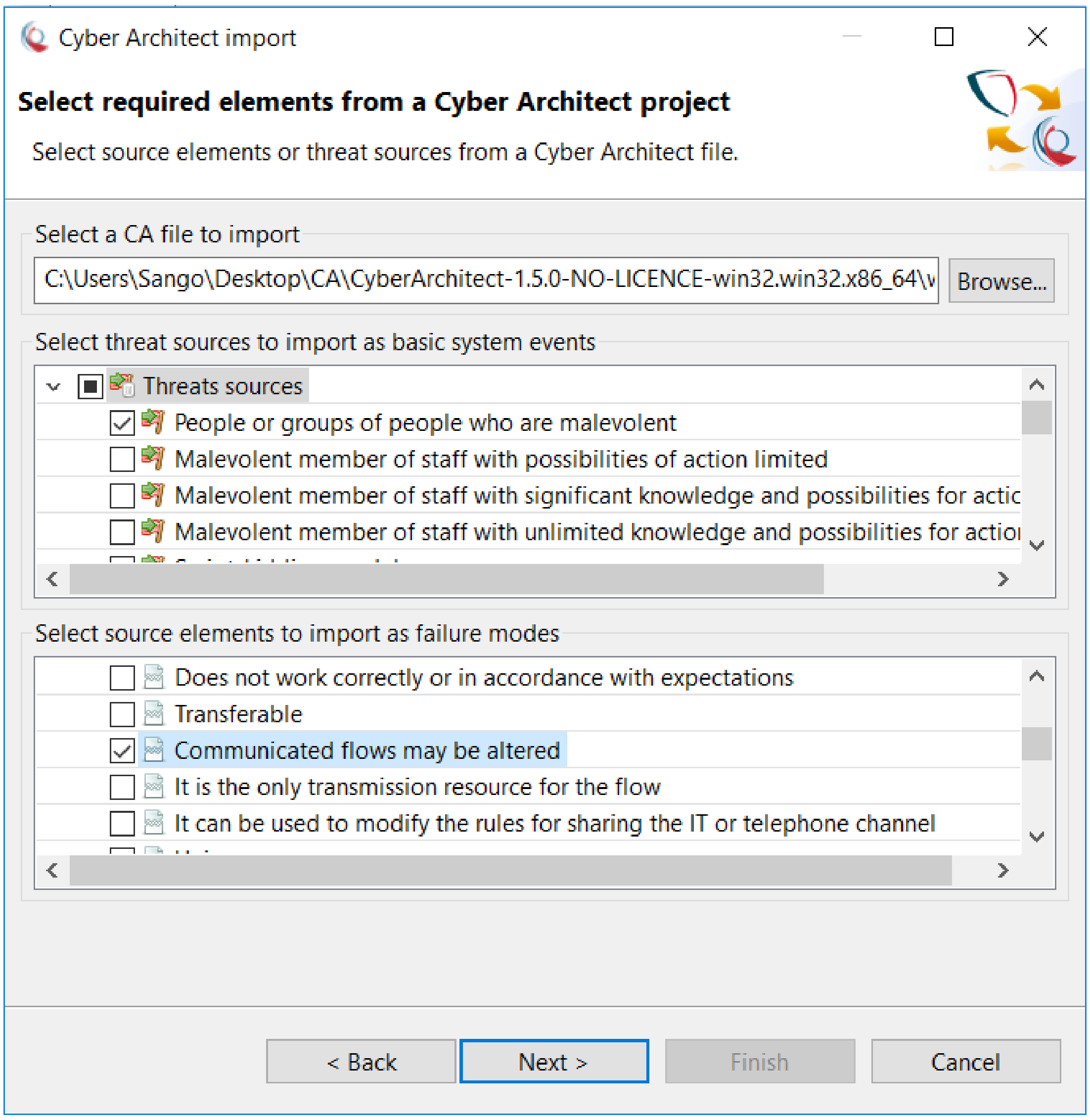


Version	Date
V1.0	10/1/18 11:56 AM

	✓ √/X	Name	Supporting asset types	Supporting assets	
28	✓	Modifiable	CHA SW PER NTW FILE		Can be improved, co
29	✓	Insufficient control by the developers or maintainers	SW		Incomplete specificat
30	✓	Does not work correctly or in accordance with expectations	SW		
31	✓	Transferable	SW		Clause of total transf
32	✓	Communicated flows may be altered	CHA NTW		Interception then re-
33	✓	It is the only transmission resource for the flow	CHA NTW		
34	✓	It can be used to modify the rules for sharing the IT or telephone channel	NTW		Transmission protoco
35	✓	Unique	CHA		

Figure 115. Cyber Architect project initialised with EBIOS knowledge bases

Step 4: Assurance Engineers (Safety & Security experts), can exploit the bridge between Safety Architect and Cyber Architect to perform it co-analysis. For example, to analyse the impact of security into safety, assurance engineers can import threats sources or vulnerabilities from Cyber Architect into Safety Architect, as illustrated in Figure 116.


Cyber Architect import

Select required elements from a Cyber Architect project

Select source elements or threat sources from a Cyber Architect file.

Select a CA file to import

C:\Users\Sango\Desktop\CA\CyberArchitect-1.5.0-NO-LICENCE-win32.win32.x86_64\v

Select threat sources to import as basic system events

- [■] Threats sources
  - [✓] People or groups of people who are malevolent
  - [ ] Malevolent member of staff with possibilities of action limited
  - [ ] Malevolent member of staff with significant knowledge and possibilities for actic
  - [ ] Malevolent member of staff with unlimited knowledge and possibilities for actio

Select source elements to import as failure modes

- [ ] Does not work correctly or in accordance with expectations
- [ ] Transferable
- [✓] Communicated flows may be altered
- [ ] It is the only transmission resource for the flow
- [ ] It can be used to modify the rules for sharing the IT or telephone channel

Figure 116. An interface between Safety Architect and Cyber Architect

Step 5: Assurance Engineers can activate the Security Viewpoint in Safety Architect tool to perform Safety and Security Co-analysis. The activation of Safety & Security viewpoint in Safety Architect allows the annotation of input and output ports of system components with vulnerability modes (e.g., communicated flows may be altered) imported in previous step. The co-analysis is realized thanks to security analysis artefacts (e.g., vulnerabilities and threats sources) and safety analysis artefacts (e.g., internal failure, failure modes), as illustrated in Figure 117.

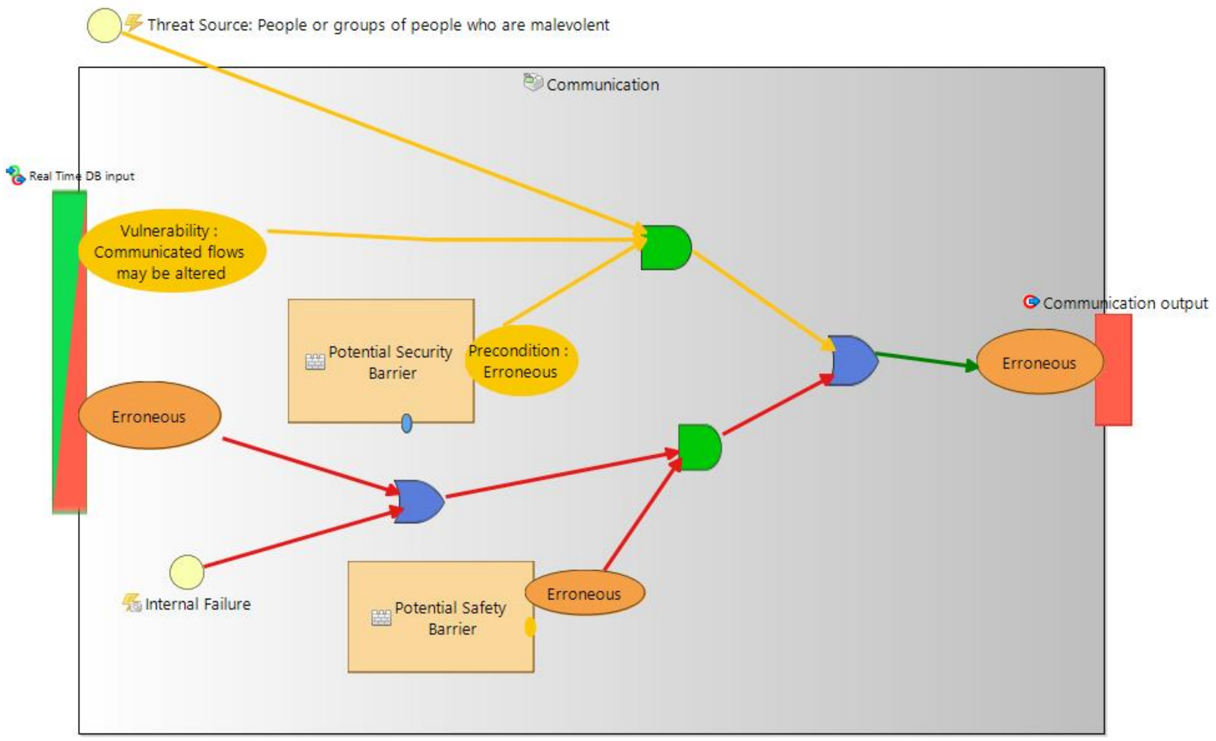
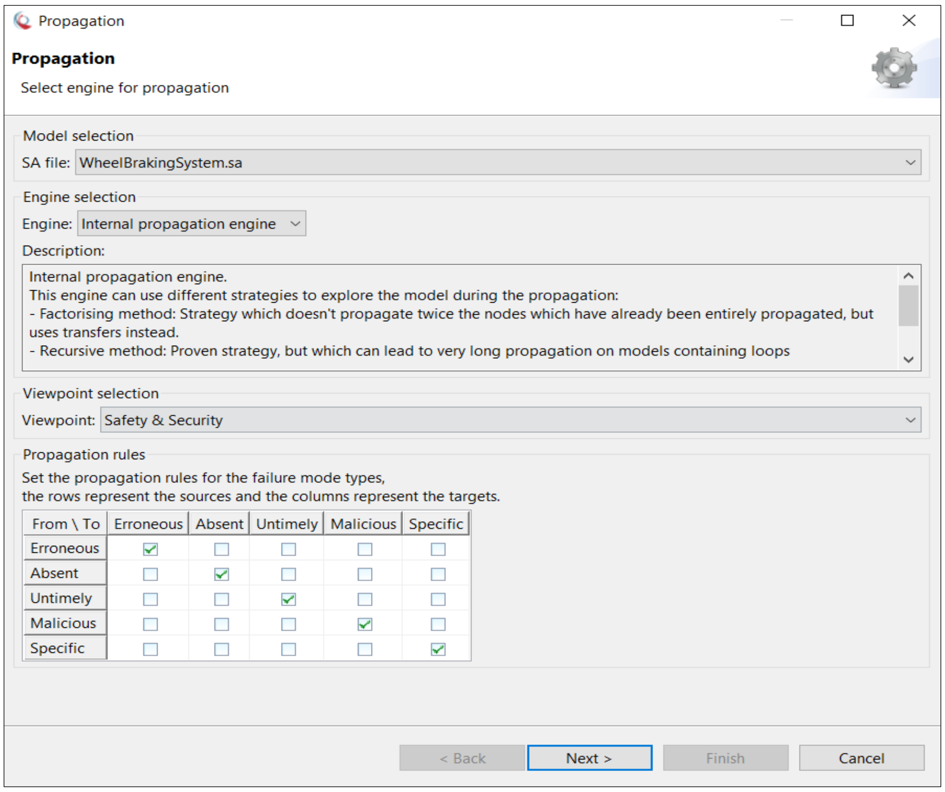


Figure 117. Safety & Security viewpoint in Safety Architect

Step 6: Assurance Engineers can generate propagation trees, i.e., fault trees extended with malicious events, thanks to the previous Safety & Security co-analysis and the Safety Architect propagation engine with the "Safety & Security" viewpoint selection shown in Figure 118.


Propagation

Propagation

Select engine for propagation

Model selection

SA file: WheelBrakingSystem.sa

Engine selection

Engine: Internal propagation engine

Description:

Internal propagation engine.
This engine can use different strategies to explore the model during the propagation:
- Factorising method: Strategy which doesn't propagate twice the nodes which have already been entirely propagated, but uses transfers instead.
- Recursive method: Proven strategy, but which can lead to very long propagation on models containing loops

Viewpoint selection

Viewpoint: Safety & Security

Propagation rules

Set the propagation rules for the failure mode types,
the rows represent the sources and the columns represent the targets.

From \ To	Erroneous	Absent	Untimely	Malicious	Specific
Erroneous	✓	☐	☐	☐	☐
Absent	☐	✓	☐	☐	☐
Untimely	☐	☐	✓	☐	☐
Malicious	☐	☐	☐	✓	☐
Specific	☐	☐	☐	☐	✓

Figure 118. Safety & Security viewpoint selection in Safety Architect

The propagation tree generated in Safety Architect can be exported in OpenPSA format (.xml files) and can be read by OpenPSA based tools, such as Arbre Analyste, as illustrated in Figure 119.

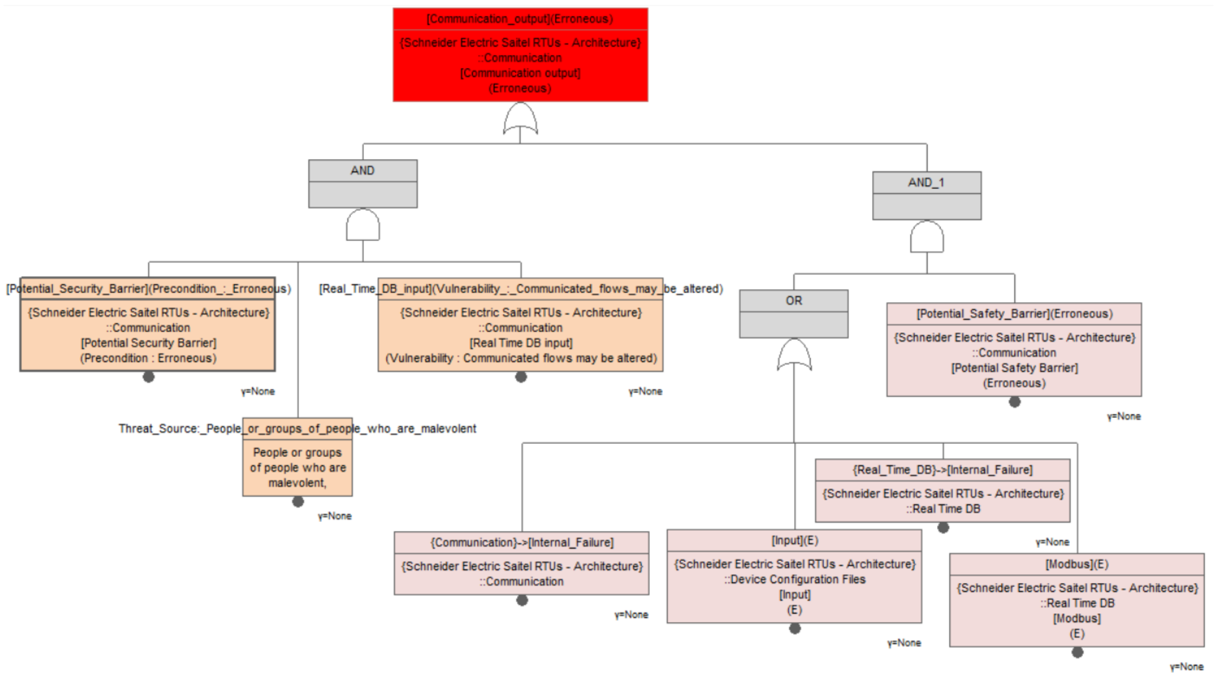

[Communication_output](Erroneous)
{Schneider Electric Saitel RTUs - Architecture} ::Communication [Communication output] (Erroneous)
AND
AND_1
[Potential_Security_Barrier](Precondition_:_Erroneous)
{Schneider Electric Saitel RTUs - Architecture} ::Communication [Potential Security Barrier] (Precondition : Erroneous)
γ=None
[Real_Time_DB_input](Vulnerability_:_Communicated_flows_may_be_altered)
{Schneider Electric Saitel RTUs - Architecture} ::Communication [Real Time DB input] (Vulnerability : Communicated flows may be altered)
γ=None
OR
[Potential_Safety_Barrier](Erroneous)
{Schneider Electric Saitel RTUs - Architecture} ::Communication [Potential Safety Barrier] (Erroneous)
γ=None
Threat_Source:_People_or_groups_of_people_who_are_malevolent
People or groups of people who are malevolent,
γ=None
{Real_Time_DB}->[Internal_Failure]
{Schneider Electric Saitel RTUs - Architecture} ::Real Time DB
γ=None
{Communication}->[Internal_Failure]
{Schneider Electric Saitel RTUs - Architecture} ::Communication
γ=None
[Input](E)
{Schneider Electric Saitel RTUs - Architecture} ::Device Configuration Files [Input] (E)
γ=None
[Modbus](E)
{Schneider Electric Saitel RTUs - Architecture} ::Real Time DB [Modbus] (E)
γ=None

Figure 119. Propagation Tree (fault tree extended with malicious events) in Safety Architect

Step 7: System Architect can import the previous propagation from Safety Architect to CHESS to display the propagation tree as a critical path in its architecture. For this, go in AMASS Platform (the eclipse bundle) then in "CHESS – Fault Tree Viewer – view fault tree diagram from .xml file", as illustrated in Figure 120.

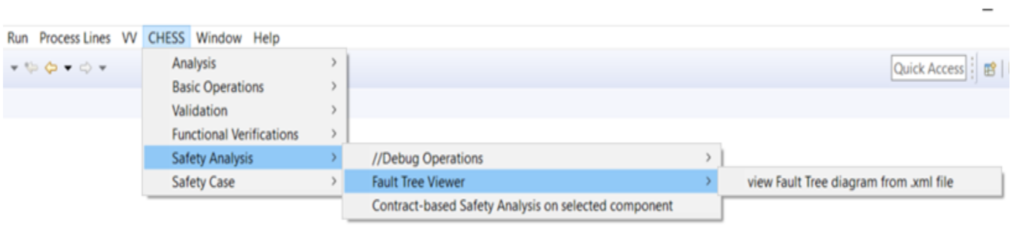
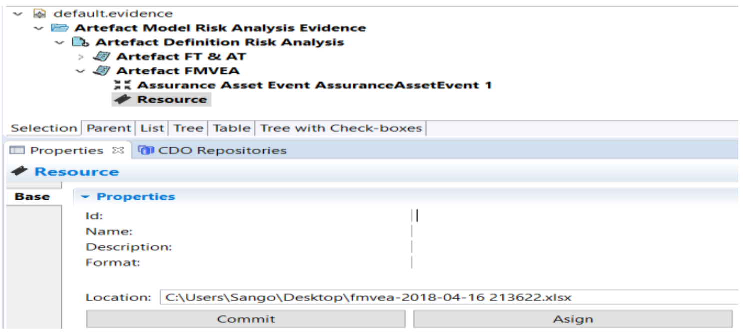


Figure 120. Import Safety Architect propagation tree in CHESS tool

Step 8: Assurance engineer can indicate the location of the evidence resource in OpenCert, such as Fault/Attack Trees or FMEA/FMVEA tables generated in Safety Architect tool, as illustrated in Figure 121.


default.evidence
Artefact Model Risk Analysis Evidence
Artefact Definition Risk Analysis
Artefact FT & AT
Artefact FMVEA
Assurance Asset Event AssuranceAssetEvent 1
Resource

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

Properties | CDO Repositories

Resource

Base | Properties

Id:
Name:
Description:
Format:
Location: C:\Users\Sango\Desktop\fmvea-2018-04-16 213622.xlsx

Commit | Asign

Figure 121. Evidence resource location in OpenCert

5. Conclusions

The Multiconcern Assurance approach of AMASS aims at capturing the multi-faceted nature of assurance with a variety of techniques that provides multi-faceted evidence (e.g. co-analysis, co-assessment) and argument fragments (multi-concern assurance) to the assurance case.

In this document, the guide for the AMASS Multiconcern Assurance approach has been given. More specifically, a set of workflows has been specified that indicates the activities to be conducted to use the AMASS Multiconcern Assurance Approach. Some case studies have been used to exemplify the execution of the workflows.

This version of the guide is related to the third and final prototype of the AMASS platform, called P2. It provides the sustainable basis for efficient, partly automated, model-based multiconcern assurance in compliance with applicable standards.

Abbreviations and Definitions

Abbreviation	*Explanation*
ACS	Attitude Control System
AOCS	Attitude and Orbit Control System
ARP	Aerospace Recommended Practice
ARTEMIS	ARTEMIS Industry Association is the association for actors in Embedded Intelligent Systems within Europe
ASIL	Automotive Safety Integrity Level
AT	Attack Tree
ATA	Attack Tree Analysis
BCL	Basic Constraint Language
BPMN	Business Process Model and Notation
BVR	Base Variability Resolution - a domain-specific language designed specifically to enable software product-line engineering (SPLE)
CA	Cyber Architect
CACC	Cooperative Adaptive Cruise Control
CACM	Common Assurance and Certification Metamodel
CCL	Common Certification Language
CHESSML	CHESS Modelling Language
CNIL	Commission Nationale de l'Informatique et des Libertés
CPS	Cyber-Physical Systems
CS	Case Study
CVL	Common Variability Language
DOORS	Dynamic Object-Oriented Requirements System
DPIA	Data Protection Impact Assessments
EBIOS	Expression des Besoins et Identification des Objectifs de Sécurité
ECSEL	Electronic Components and Systems for European Leadership
ECSS	European Cooperation for Space Standardization
EMC^2	Embedded multi-core systems for mixed criticality applications in dynamic and changeable real-time environments
EPF-C	Eclipse Process Framework-Composer
EU	European Union
FLAMM	Failure Logic Analysis Meta Model
FMEA	Failure Modes and Effects Analysis
FMVEA	Failure Modes, Vulnerabilities and Effect Analysis
FODA	Feature-Oriented Domain Analysis
FT	Fault Tolerance
FPTC	Failure Propagation Transformation Calculus
FTA	Fault Tree Analysis
GDPR	General Data Protection Regulation
GSN	Goal Structured Notation
GUI	Graphical User Interface
HARA	Hazard Analysis and Risk Assessment

HAZOP	HAZard and OPerability study
HW	Hardware
IACS	Industrial and Automation Control Systems
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
IT	Information Technology
JU	Joint Undertaking
LTL	Linear-time Temporal Logic
MARTE	Modelling and Analysis of Real Time and Embedded systems
MAST	Modelling and Analysis Suite for Real-Time Applications
MERgE	Multi-Concerns Interactions System Engineering
MOF	Meta-Object Facility
NIST	National Institute of Standards and Technology
OCRA	Othello Contracts Refinement Analysis
OPENCOSS	Open Platform for EvolutioNary Certification of Safety-critical Systems
ReqIF	Requirements Interchange Format
RCP	Rich Client Format
RobotML	Robot Modelling Language
RTU	Remote Terminal Unit
SA	Safety Architect
SACM	Structured Assurance Case Metamodel
SAE	Society of Automotive Engineers
SAHARA	Security-aware Hazard Analysis and Risk Assessment
SIL	Safety Integrity Level
SL	Security Level
SiSoPLE	Security-informed Safety-oriented Process Line Engineering
SoPLE	Safety-oriented Process Line Engineering
SPEM	Software & Systems Process Engineering Metamodel
SPLCA	Software Product Line Covering Array
SSA	System Safety Assessment
SSE	Safety and Security Engineering
STL	Signal Temporal Logic
STO	Scientific Technical Objective
STPA-SEC	STAMP (Systems- Theoretic Accident Model and Processes) Based Process Analysis
STRIDE	Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege
SUDI	Software Unit Design and Implementation
SUT	System Under Test
SVN	Subversion
SW	Software
SysML	System Modelling Language
TARA	Threat Analysis and Risk Assessment
UMA	Unified Method Architecture
UML	Unified Modelling Language

URL	Uniform Resource Locator
V&V	Verification and Validation
WBS	Work Break Down Structure
WEFACT	Workflow Engine for Analysis, Certification and Test
WP	Work Package
XML	EXtensible Markup Language
xSAP	Extended Safety Analysis Platform

References

[1] AMASS D1.1 Case studies description and business impact, 30th November 2016.

[2] AMASS D4.1 Baseline and requirements for multiconcern assurance, 30th September 2016.

[3] AMASS D4.3 Design of the AMASS tools and methods for multiconcern assurance (b), 30th April 2018

[4] AMASS D4.5 Prototype for multiconcern assurance (b), 31th October 2017.

[5] Goal Structuring Notation Working Group. GSN Community Standard. Retrieved from http://www.goalstructuringnotation.info, Nov 2011.

[6] AMASS D3.3 Design of the AMASS tools and methods for architecture-driven assurance (b), 31st March 2018.

[7] AMASS D3.8 Methodological guide for architecture-driven assurance (b), 31th October 2018.

[8] AMASS D6.8 Methodological guide for cross/intra-domain reuse (b), 31th October 2018.

[9] AMASS D5.3 Design of the AMASS tools and methods for seamless interoperability (b), 30th June 2018.

[10] AMASS D6.3 Design of the AMASS tools and methods for cross/intra-domain reuse (b), 31st July 2018.

[11] AMASS D4.6 Prototype for multiconcern assurance (c), 31st August 2018.

[12] AMASS D2.5 AMASS user guidance and methodological framework, October 2018.

[13] P. Koopman. Better Embedded System Software. Drumnadrochit Education LLC, ISBN-13: 978-0-9844490-0-2, 2010.

[14] https://www.all4tec.net/safety-architect

[15] https://www.all4tec.com/cyber-architect

[16] MERgE Project – http://www.merge-project.eu/

[17] https://www.ssi.gouv.fr/guide/ebios-2010-expression-des-besoins-et-identification-des-objectifs-de-securite/

[18] Eclipse Process Framework Project. http://www.eclipse.org/epf/

[19] Object Management Group: The Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0 (2008). http://www.omg.org/spec/SPEM/2.0/

[20] BVR Tool. https://github.com/SINTEF-9012/bvr

[21] ISO 26262: Road Vehicles-Functional Safety. International Standard, 2011.

[22] SAE J3061: Surface Vehicle Recommended Practice, Cybersecurity Guidebook for Cyber-Physical Vehicle Systems, Tech. Rep., January 2016.

[23] Eclipse Process Framework (EPF) Composer User Manual.
https://www.eclipse.org/epf/general/EPF_Installation_Tutorial_User_Manual.pdf

[24] BVR Tool. https://bvr-tool.sintef.cloud/

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

[26] The OPENCOSS project (Open Platform for EvolutioNary Certification Of Safety-critical Systems) http://www.opencoss-project.eu/

[27] The CHESS tool https://www.polarsys.org/projects/polarsys.chess

[28] Arbre Analyste https://www.arbre-analyste.fr

[29] J. P. Castellanos Ardila and B. Gallina. Towards Efficiently Checking Compliance Against Automotive Security and Safety Standards. Proceedings of the 7nd IEEE WoSoCER, joint event of the 28th International Symposium on Software Reliability (ISSRE), IEEE Computer Society, Toulouse, France, 23 of October 2017.

[30] B. Gallina, S. Kashiyarandi, H. Martin, R. Bramberger.Modeling a safety-and automotive-oriented process line to enable reuse and flexible process derivation. In Proceedings of the IEEE 38th

International Computer Software and Applications Conference Workshops (COMPSACW), pp. 504-509, IEEE, 2014.

[31] C. Schmittner, Z. Ma, P. Puschner. Limitation and Improvement of STPA-Sec for Safety and Security Co-analysis. In International Conference on Computer Safety, Reliability, and Security (SAFECOMP, pp. 195-209, Springer International Publishing, September 2016.

[32] C. Schmittner, Z. Ma, P. Smith. FMVEA for safety and security analysis of intelligent and cooperative vehicles. In International Conference on Computer Safety, Reliability, and Security (SAFECOMP), pp. 282-288, Springer, Cham, September 2014.

[33] A. Ruiz Lopez, B. Gallina, J. Luis de la Vara, S. Mazzini, H. Espinoza Ortiz. AMASS: Architecture-driven, Multi-concern, Seamless, Reuse-Oriented Assurance and Certification of CPSs. 5th International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems (SASSUR), Trondheim, Norway, September 2016.

[34] B. Gallina, Z. Haider, A. Carlsson. Towards Generating ECSS-compliant Fault Tree Analysis' Results via ConcertoFLA. 2nd International Conference on Reliability Engineering (ICRE), Milan, Italy, December 20-22, 2017.

[35] B. Gallina, E. Sefer, and A. Refsdal. Towards Safety Risk Assessment of Socio-technical Systems via Failure Logic Analysis. 2nd IEEE International Workshop on Risk Assessment and Risk-driven Testing (RISK), joint event of ISSRE, doi: 10.1109/ISSREW.2014.49, pp.287-292, Naples, Italy, November 3-6, 2014.

[36] Mazzini, S., Favaro, J., Puri, S., Baracchi, L.: CHESS: an open source methodology and toolset for the development of critical systems. Third Workshop on Open Source Software for Model Driven Engineering (OSS4MDE 2016).

[37] WEFACT user manual and Installation Instructions, 2018[7].

[38] International Electro-technical Commission, *"Functional safety of electrical/ electronic/ programmable electronic safety-related systems"*, IEC-61508, 1998.

[39] International Organization for Standardization, *"Road vehicles — Functional safety"*, ISO-26262, 2011.

[40] International Organization for Standardization, *"Robots and robotic devices - Safety requirements for personal care robots"*, ISO-13482, 2014.

[41] SAE International, *"Guidelines for Development of Civil Aircraft and Systems"*, Aerospace Recommended Practice, ARP-4754, 1996.

[42] SAE International, *"Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment"*, Aerospace Recommended Practice, ARP-4761, 1996.

[43] International Organization for Standardization / International Electro-technical Commission, *"Information Technology – Security Techniques – Information Security Risk Management"*, ISO/IEC 27005, 2013.

[44] International Organization for Standardization / International Electro-technical Commission, *"Information technology - Security techniques - Evaluation criteria for IT security"*, ISO/IEC-15408, 2009.

[45] International Electro-technical Commission, *"Industrial communication networks - Network and system security"*, ISA/IEC-62443, 2013.

[46] The European Organisation for Civil Aviation Equipment, *"Airworthiness Security Process Specification"*, ED-202A, 2014.

[47] The European Organisation for Civil Aviation Equipment, *"Airworthiness Security Methods and Considerations"*, ED-203, 2015.

[48] Radio Technical Commission for Aeronautics, *"Airworthiness Security Methods and Considerations"*, RTCA DO-356, 2014.

[7] Available in the AMASS project repository: WEFACT_UserManual.docx and WEFACT_Installation_Instructions.docx

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

[50] ECSS-E-ST-40C, Space engineering - Software, 06/03/2009.

[51] ECSS-Q-ST-80C, Space product assurance - Software product assurance, 06/03/2009.

[52] ECSS-Q-ST-30C, Space product assurance - Dependability, 06/03/2009.

[53] ECSS-Q-ST-40C, Space product assurance - Safety, 06/03/2009.

[54] G. Macher, A. Hoeller, H. Sporer, E. Armengaud, C. Kreiner. A Comprehensive Safety, Security, and Serviceability Assessment Method. 34th International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Delft, The Netherlands, 2015.

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

[56] T. Amorim, H. Martin, Z. Ma, Ch. Schmittner, D. Schneider, G. Macher, B. Winkler, M. Krammer, Ch. Kreiner. Systematic Pattern Approach for Safety and Security Co-engineering in the Automotive Domain. In Proceedings of the 36th International Conference on Computer Safety, Reliability, and Security (SAFECOMP), Lecture Notes in Computer Science, vol 10488. Springer, Cham, Trento, Italy, 2017.

[57] OMG, Structured Assurance Case Metamodel (SACM) version 2 http://www.omg.org/spec/SACM/2.0/Beta1/ , June 2016

[58] The Hon. Lord Cullen (1990), The Public Inquiry into the Piper Alpha Disaster, Vols. 1 and 2 (Report to Parliament by the Secretary of State for Energy by Command of Her Majesty, November).

[59] http://www.arrowhead.eu

[60] E. Denney, G. Pai and I. Habli. Dynamic Safety Cases for Through-Life Safety Assurance. IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE), Florence, pp. 587-590, doi: 10.1109/ICSE.2015.199, 2015.

[61] Ø. Haugen and O. Øgård. BVR - better variability results. Proceedings of the 8th International Conference on System Analysis and Modelling: Models and Reusability (SAM), Valencia, Spain. In D. Amyot, P. Fonseca i Casas, and G. Mussbacher, editors, System Analysis and Modelling: Models and Reusability, volume 8769 of Lecture Notes in Computer Science, pages 1–15, Springer International Publishing, 2014.

[62] Ø. Haugen, “Common Variability Language (CVL),” Object Management Group, Tech. Rep. ad/2012-08-05, August 2012.

[63] VARIES D4.2- BVR - The language. https://bvr-tool.sintef.cloud/docs/VARIES_D4.2_v01_PP_FINAL.pdf

[64] VARIES, http://www.varies.eu/, accessed: 2017-07-13.

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

[66] M. F. Johansen, O. Haugen, F. Fleurey, A. G. Eldegard, and T. Syversen. Generating better partial covering arrays by modelling weights on sub-product lines. Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MODELS), Innsbruck, Austria, LNCS, vol 7590. Springer, pp. 269-284, September 30-October 5, 2012.

[67] A. Vasilevskiy and Ø. Haugen. Resolution of interfering product fragments in software product line engineering. In J. Dingel, W. Schulte, I. Ramos, S. Abrahão, and E. Insfran, editors, Model-Driven Engineering Languages and Systems, volume 8767 of Lecture Notes in Computer Science, pages 467–483. Springer International Publishing, 2014.

[68] Eclipse Process Framework Composer, Part 1: Key Concepts, by Peter Haumer, 2007. https://eclipse.org/epf/general/EPFComposerOverviewPart1.pdf

[69] Eclipse Process Framework Composer, Part 2: Authoring method content and processes, by Peter

Haumer, 2007. http://www.eclipse.org/epf/general/EPFComposerOverviewPart2.pdf

[70] International Organization for Standardization / International Electro-technical Commission, “Analysis techniques for system reliability – Procedure for failure mode and effects analysis (FMEA)”, IEC 61812, 2006.

[71] ESSB-ST-E-008 - Secure Software Engineering Standard, 2016.

[72] Z. Haider, B. Gallina and E. Zornoza “FLA2FT: Automatic generation of fault tree from ConcertoFLA results” 3rd International Conference on System Reliability and Safety (ICSRS), 2018.

[73] Ewen Denney, Ganesh Pai and Ibrahim Habli. “Dynamic Safety Cases for Through-life Safety Assurance” In 37th International Conference on Software Engineering (ICSE. 2015)–New Ideas and Emerging Results (NIER) 2015 May

[74] OPENCOSS D5.6 Compositional Certification Framework: Methodological Guide (report), March 2015

[75] Marco Gario, Alessandro Cimatti, Cristian Mattarei, Stefano Tonetta, Kristin Yvonne Rozier: Model Checking at Scale: Automated Air Traffic Control Design Space Exploration. CAV (2) 2016: 3-22

[76] Ieuan Jolly, Data protection in the United States: overview, in Data Protection Global Guide, Thomson Reuters, 2017

[77] European Parliament and Council, Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance), in Official Journal of the European Union, 2016

[78] Commission de la Protection de la Vie Privée (CPVP), Recommandation n° 01/2018, 2018

[79] Article 29 Data Protection Working Party (WP29), Guidelines on Data Protection Impact Assessment (DPIA) and determining whether processing is “likely to result in a high risk” for the purposes of Regulation 2016/679, 2017

[80] Commission Nationale de l'Informatique et des Libertés (CNIL), Privacy Impact Assessment (PIA) Knowledge Bases, 2018

[81] National Institute of Standards and Technology (NIST), Special Publication 800-53 - Security and Privacy Controls for Federal Information Systems and Organizations, 2013

[82] Lin Liu, Eric Yu, John Mylopoulos, Security and Privacy Requirements Analysis within a Social Setting, in Proceedings of the 11th IEEE International Requirements Engineering Conference, 2003

[83] Mina Deng, Kim Wuyts, Riccardo Scandariato, Bart Preneel, Wouter Joosen, A privacy threat analysis framework: supporting the elicitation and fulfilment of privacy requirements, in Journal of Requirements Engineering, 2010

[84] Katia Hayati, Martín Abadi, Language-Based Enforcement of Privacy Policies, in Privacy Enhancing Technologies, PET 2004, Lecture Notes in Computer Science, vol 3424, 2004

[85] Gergö Barany, Julien Signoles, Hybrid Information Flow Analysis for Real-World C Code, in Tests and Proofs, TAP 2017, Lecture Notes in Computer Science, vol 10375, 2017

[86] Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, Boris Yakobowski, Frama-C: A software analysis perspective, in Journal of Formal Aspects of Computing, 2015

[87] Patrick Baudin, Pascal Cuoq, Jean-Christophe Filliâtre, Claude Marché, Benjamin Monate, Yannick Moy, Virgile Prevosto, ACSL: ANSI/ISO C Specification Language, 2010

[88] The Concerto Project, http://www.concerto-project.org/

Appendix A. Changes with respect to D4.7 (*)

New Sections:

Section	*Title*
2.1.1.1.	Contract-based trade-off analysis in parameterized architectures
3.1.1	Contract-based trade-off analysis in parameterized architectures
3.4	Standard-related dependability co-assessment via OpenCert
3.6	Privacy Analysis
4.4	CS1: Industrial and Automation Control Systems (IACS)

Sections whose number has changed:

Former Section No.	*New Section No.*	*Title*
3.4	3.5	System Dependability Co-Analysis

Modified Sections:

Chapter/Section	*Title*	*Change*
1	Introduction	Minor changes to better explain the context/motivation of the final version of this deliverable.
2.1.1	Contract Based Multi-concern Assurance	Addition of 2.1.1.1
2.3.3	FMVEA	Extension
3.1	Contract-Based Multiconcern Assurance	Extension
3.2	Dependability assurance case modelling	Enhancement of the guidelines and restructuring
4.1	Case Study CS11 - Attitude and Orbit Control System	Extension of the case study and application of multi-concern analysis.