ECSEL Research and Innovation actions (RIA)

AMASS

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

# Prototype for multi-concern assurance (c)
# D4.6

| | |
|---|---|
| **Work Package:** | WP4 Multi-Concern Assurance |
| **Dissemination level:** | PU = Public |
| **Status:** | Final |
| **Date:** | 31 August 2018 |
| **Responsible partner:** | Thomas Gruber (AIT) |
| **Contact information:** | Thomas.gruber@ait.ac.at |
| **Document reference:** | AMASS_D4.6_WP4_AIT_1.0 |

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

## Contributors

| Names | Organisation |
|---|---|
| Thomas Gruber, Christoph Schmittner, Sebastian Chlup, Korbinian Christl, Siddhartha Verma, Abdelkader Shaaban | AIT Austrian Institute of Technology GmbH (AIT) |
| Alejandra Ruiz, Garazi Juez | Tecnalia Research & Innovation (TEC) |
| Barbara Gallina, Irfan Sljivo, Zulqarnain Haider | Maelardalen Hoegskola (MDH) |
| Stefano Puri | Intecs (INT) |
| Stefano Tonetta, Alberto Debiasi | Fondazione Bruno Kessler (FBK) |
| Jan Mauersperger, Nino Gabriel | ANSYS (KMT) |

## Reviewers

| Names | Organisation |
|---|---|
| Garazi Juez (Peer reviewer) | Tecnalia Research & Innovation (TEC) |
| Marc Sango (Peer reviewer) | All4Tec (A4T) |
| Barbara Gallina (TC reviewer) | Maelardalen Hoegskola (MDH) |
| Jose Luis de la Vara (TC reviewer) | Universidad Carlos III de Madrid (UC3) |
| Alejandra Ruiz (TC reviewer) | Tecnalia Research & Innovation (TEC) |
| Cristina Martinez (Quality Manager) | Tecnalia Research & Innovation (TEC) |

# TABLE OF CONTENTS

# List of Figures

# List of Tables

# Executive Summary (*)

This deliverable, D4.6 Prototype for multi-concern assurance (c), is the third output of the task T4.3 *Implementation for Multi-Concern Assurance*. Based on the results from the task T2.2 *AMASS Reference Tool Architecture and Integration*, the task T4.3 develops a prototype tooling for multi-concern assurance. Particular attention is paid to support the architectural approach to assurance being developed in WP3 and to implement the requirements for tooling aimed at supporting the multi-concern approach, developed in WP4. The task 4.3 has been carried out iteratively, in close connection with the conceptual tasks (T4.2 *Conceptual Approach for Multi-Concern Assurance* as well as those in the other WPs, namely T3.2, T5.2 and T6.2), with validation results from the implementation being used to guide further refinement of the conceptual approach. The implementation is closely guided by the requirements [38] of the case studies, which are used to validate the prototype.

The first prototype iteration (Prototype Core) released the basic building blocks as a consolidation/integration of previous projects OPENCOSS [1] and SafeCer [2]. The developed tools in the first prototype (Prototype Core) supported the following two functional areas:

- Argumentation Editor
- Argument Patterns Editor

The second prototype iteration (P1) extended the previous functionality by the following functional parts:

- Support for contract-based multi-concern assurance by CHESS, and
- Multi-concern assurance workflow support by WEFACT [21] based on
- Standards conformant assurance process modelling by EPF-C [9].

The release at hand is the third prototype iteration (P2), which extends the previous functionalities by further functional parts and adds additional external tools as listed in the following:

- Support for contract-based multi-concern assurance by CHESS.
- Further extensions to CHESS regarding Contract-based trade-off analysis in parameterized architectures.
- Concerto-FLA extension.
- Failure Mode, Vulnerabilities and Effect Analysis (FMVEA) tool.
- Analytical Network Process (ANP) tool prototype.
- MORETO tool.
- Medini Analyzer.

This document has the purpose to present the added functional parts in detail, which are partly Open Source tools integrated in the AMASS platform and partly external tools, for which the binding via an open source interface module is given.

CHESS and EPF-C are already used in other contexts of the AMASS ARTA platform; therefore, references to the comprehensive specifications elsewhere are given and a short description is included in this document pointing out the particularity of the tool in context with the WP4 task of multi-concern assurance. The BVR Tool is used in AMASS for managing the variability. Its selection and integration are part of WP6-work, where variability management for enabling systematic reuse is in focus.

In the context of WP4, the role of BVR Tool is related to managing the variability when co-engineering (cross-concern) is in focus. For this reason, it is mentioned in this deliverable as well.

The WEFACT workflow engine as an external tool was integrated via an open source interface module in iteration two and is described in detail in the document at hand. In the third iteration, specifications of the additional external tools FMVEA, MORETO, Medini Analyzer and the ANP tool prototype have been added. This includes references to the open source interface, information about the technology used and a description of the mapping between the tool-internal database and the AMASS CACM. In most cases, existing open interfaces could be used for coupling the external tools to the AMASS platform.

Important parts of D4.6 are:

- Executables of or references to the external tools WEFACT, FMVEA, MORETO, Medini Analyzer and the ANP tool prototype,
- User manuals and installation instructions, and
- where applicable, source code of the interface modules (e.g. in [18]).

Pointers to these parts are intended to be provided with D2.5 [45].

This deliverable represents an update of AMASS D4.5 [51] which was released in m19; the sections modified with respect to D4.5 have been marked with (*) in the headlines, those which are new with (**).

# 1. Introduction (*)

The AMASS approach focuses on the development and consolidation of an open and holistic assurance and certification framework for CPS, which constitutes the evolution of the OPENCOSS [1] and SafeCer [2] approaches towards an architecture-driven, multi-concern assurance, reuse-oriented, and seamlessly interoperable tool platform.

The expected tangible AMASS results are:

a) The **AMASS Reference Tool Architecture**, which extends the OPENCOSS and SafeCer conceptual, modelling and methodological frameworks for architecture-driven and multi-concern assurance, as well as for further cross-domain and intra-domain reuse capabilities and seamless interoperability mechanisms (based on OSLC specifications [14]).

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

c) The **Open AMASS Community**, which will manage the project outcomes, for maintenance, evolution and industrialisation. The Open Community will be supported by a governance board, and by rules, policies, and quality models. This includes support for AMASS base tools (tool infrastructure for database and access management, among others) and extension tools (enriching AMASS functionality). As Eclipse Foundation is part of the AMASS consortium, the Polarsys/Eclipse community (www.polarsys.org) is a strong candidate to host AMASS Open Tool Platform.

To achieve the AMASS results, as depicted in Figure 1, the multiple challenges and corresponding scientific and technical project objectives are addressed by different work-packages.

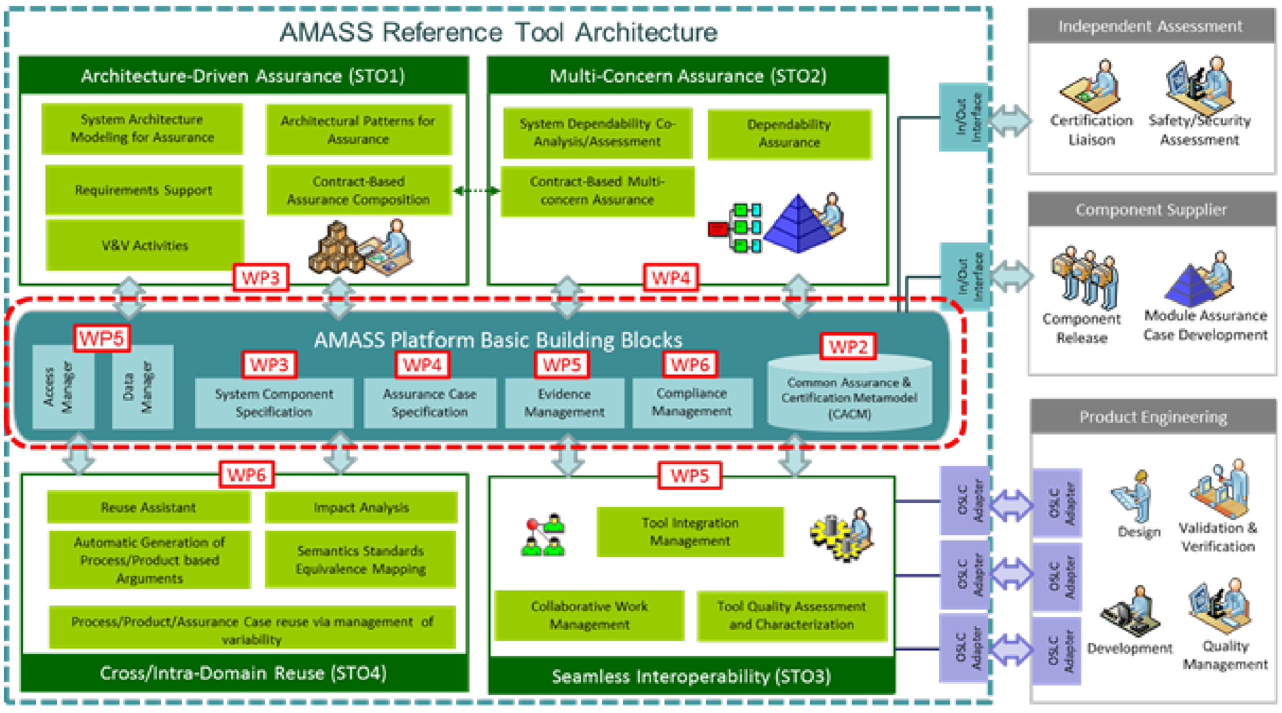

AMASS Reference Tool Architecture
Architecture-Driven Assurance (STO1)
System Architecture Modeling for Assurance
Architectural Patterns for Assurance
Requirements Support
Contract-Based Assurance Composition
V&V Activities
WP3
Multi-Concern Assurance (STO2)
System Dependability Co-Analysis/Assessment
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
Independent Assessment
Certification Liaison
Safety/Security Assessment
Component Supplier
Component Release
Module Assurance Case Development
Product Engineering
Design
Validation & Verification
Development
Quality Management

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

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

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

AMASS has provided three prototype iterations:

1. During the **first prototyping** iteration (Prototype Core), the AMASS Platform Basic Building Blocks (see Figure 1), were aligned, merged and consolidated at TRL4[1].
2. During the **second prototyping** iteration (Prototype P1), the AMASS-specific Building Blocks were developed and benchmarked at TRL4; this comprises the blue basic building blocks as well as the green building blocks in Figure 1. Regarding multi-concern assurance, in this second prototype, the specific building blocks provide functionalities regarding system dependability co-analysis/assessment, dependability assurance modelling or contract-based multi-concern assurance.
3. Finally, at the **third prototyping** iteration (Prototype P2), all AMASS building blocks have been integrated in a comprehensive toolset operating at TRL5. Functionalities specific for multi-concern assurance developed for the second prototype were improved and integrated with functionalities from other technical work packages.

Each of these iterations has the following three prototyping dimensions:

- ***Conceptual/research development***: development of solutions from a conceptual perspective.
- ***Tool development***: development of tools implementing conceptual solutions.
- ***Case study development***: development of industrial case studies using the tool-supported solutions. The application of the building blocks in the case studies for the first prototype was described in D1.1 [22], for the second prototype in D1.5 [46]. In the third iteration P2, implementations applying WEFACT and FMVEA are under elaboration in CS1 and CS3, MORETO is used in CS1, Medini Analyzer in CS3. The application of Concerto-FLA is contained in CS4, and CHESS with OCRA is applied for contract-based multi-concern assurance in CS1, CS5, CS9 and CS10. Finally, the OpenCert Assurance Case Editor is used in more than half of the case studies.

As part of the Prototype Core, WP4 provided the implementation of the basic building block "**Assurance Case Specification**" (Figure 1). An update of the respective Assurance Case Editor is given in section 3.1.

This deliverable reports the **tool and interface module development** of the "Multi-concern Assurance" building blocks and explains the final implementation. This refers to the following functionalities:

- Support for contract-based multi-concern assurance and for trade-off analysis based on parameterized architectures by the internal tool CHESS,
- Standards conformant assurance process modelling by the internal tool EPF-C, and
- Multi-concern assurance workflow supporting combined activity execution for different multi-concern assurance functions by means of the external tool WEFACT,
- Safety-security co-analysis by the external tool FMVEA,
- Failure-Logic Analysis with the internal tool Concerto-FLA,
- Trade-off analysis with the prototypic external ANP (Analytical Network Process) tool,
- Security analysis and requirements allocation with the external tool MORETO, and
- Safety-security co-analysis by the external tool Medini Analyzer.

[1] In the context of AMASS, the EU H2020 definition of TRL is used, see https://ec.europa.eu/research/participants/data/ref/h2020/other/wp/2016_2017/annexes/h2020-wp1617-annex-g-trl_en.pdf

With respect to the EPF-C and CHESS tools, this deliverable contains short descriptions and refers to other deliverables in which the mentioned tools are already described for a different context. For the external tool WEFACT, which is integrated via the mentioned interface module, this deliverable presents the WP4 functionality and describes the interface and its mapping to the CACM in detail. The WP6-related functions for process-based argument generation are mentioned only shortly and a reference to the descriptions in WP6 are given.

Other important parts of this deliverable are:

- Installable AMASS Platform tools or open-source interface module for the third prototype,
- User Manuals and installation instructions, and
- Source code description.

# 2. Implemented Functionality (*)

## 2.1 Scope (*)

This third prototype of the Multi-concern assurance module has the purpose to extend the functionality of the second prototype by additional developments and to provide interface modules yet missing in the second iteration (P1). It completes the full scope of multi-concern assurance-related functions with internal and external tools.

The following tool functions were already integrated in the first iteration of the AMASS platform:

- OpenCert – AMASS Core edition supporting (only) "Assurance case specification", and
- CHESS - AMASS Core edition supporting contract modelling with OCRA.

In the second iteration of the AMASS platform (prototype P1), the following tools were integrated or extended with respect to functionality:

1. OpenCert – AMASS P1 edition supports, in addition to "Assurance Case Specification":
    - "Dependability Assurance Modelling", and
    - partly "Contract–based multi-concern assurance"
2. CHESS - supports additionally "Contract-Based Multi-concern Assurance"
3. EPF-Composer – supports:
    - "Co-assessment, Cross-Concern Reuse" (shared with WP6, the process model is made vary with respect to the desired concern by BVR Tool), and
    - Assurance process modelling and tailoring to the individual project (resulting process model is used by WEFACT)
4. WEFACT - supports the assurance process workflow (this concerns several WPs).
    - In WP4, the capability of combining analysis tools, targeting different concerns, is in focus.

The following tools, described in deliverable D4.3 [25], have been integrated in the third iteration of the AMASS platform (prototype P2):

- Further extensions to CHESS regarding Contract-based trade-off analysis in parameterized architectures and support for contract-based multi-concern assurance,
- Concerto-FLA – Extension of the Concerto-FLA tool (see [33], [34]) allowing Failure Logic Analysis (FLA) not only for safety but also for security-related failure modes,
- FMVEA tool – supports model-based system-dependability co-analysis and –assessment,
- ANP (Analytical Network Process) tool prototype – supports trade-off analyses between various quality attributes based on an ANP using coloured Petri Nets,
- MORETO - supports security analysis and manual or standards-based automated generation of security requirements,
- Medini Analyzer - supports the assurance process workflow and allows safety and security analyses.

A few tools were mentioned in earlier iterations of this deliverable as potential candidates for the integration with the AMASS platform as WP4 functionalities, but finally the following decisions were taken:

The Farkle tool, which verifies learning algorithms based on volume testing, supports product assurance for a very specific case; its use is being investigated but not planned to be integrated with the AMASS platform.

The AMT2.0 (Analogue Monitoring Tool), which supports "Contract-Based Multi-concern Assurance" by generating monitors for observing properties of nodes a network, has been identified as a tool supporting architecture-based assurance and is therefore now described in WP3.

## 2.2 Implemented Requirements - Overview (*)

The WP4 tools contained in the final iteration P2 of the AMASS platform provide solutions for a set of AMASS requirements as defined in deliverable D2.1 [16]. Apart from the WP4 requirements, these tools fulfil also several requirements related to other work packages. The following Table 1 lists all these requirements including the WP4 tools that fulfil them.

**Table 1.** Requirements implemented in the third prototype of the AMASS platform (P2)

| Requirement No | Name | Description | Tool |
|---|---|---|---|
| **WP4_ACS_001** | Assurance case edition | The system shall be able to edit an assurance case in a scalable way. | OpenCert |
| **WP4_ACS_002** | Argumentation architecture | The system shall be able to edit a modular structure (argument architecture) associated with a system and/or component. | OpenCert |
| **WP4_ACS_003 (Core implementation improved)** | Drag and drop argumentation patterns | The system shall be able to instantiate in the actual assurance case an argument pattern (concerning safety and security) selected from the list of patterns stored. | OpenCert |
| **WP4_ACS_005** | Provide a structured language to the text inside the claims | The system could be able to provide support for language formalization inside argument claims. | OpenCert |
| **WP4_ACS_006** | Provide guidelines for argumentation | The system could be able to provide guidelines about the assurance case edition based on the system/component development phase status. | OpenCert, WEFACT in a specific way[2]) |
| **WP4_ACS_007** | Argumentation import/export | The system could be able to import/export argumentations to SACM [5]. | OpenCert |
| **WP4_ACS_008** | Traceability of the dependability case | The system should provide the dependability case reviewers the ability of tracing an overall dependability case (GSN) goal to the requirement within the dependability profile for a given system element and the attribute of interest with which goal is associated. | OpenCert, WEFACT (partly) |
| **WP4_ACS_010** | Composition of the overall argument | The system should provide the capability of generating a compositional assurance case argument. | OpenCert |
| **WP4_ACS_011** | Assurance case status report | The system could provide the capability for querying the assurance case in order to detect: 1) undeveloped goals, 2) fallacies. | WEFACT |
| **WP4_ACS_013** | Provide quantitative confidence metrics about an assurance case in a report | The system could produce a status report indicating a quantitative confidence metric for assurance case. | WEFACT |
| **WP4_CAC_010** | Contract-based trade-off analysis | The system could provide the capability to evaluate safety and security requirements on different system architectures to perform trade-off analysis based on the contract specification. | ANP tool (partly), CHESS |
| **WP4_DAM_001** | Capability to model relationships between concerns | The system shall be able to provide an assurance case which records the relationships between dependability attributes and how they are affected because of design decisions. | OpenCert |
| **WP4_DAM_002** | Capability to capture conflicts occurring during system development and the | The system shall provide the capability for modelling a dependability case that captures the conflicts that occur during system development and the trade-off process to justify why the taken design decisions are | OpenCert, ANP tool |

[2] For an explanation see section 2.2.4.

| Requirement No | Name | Description | Tool |
|---|---|---|---|
| | trade-off process | the most optimal ones. | |
| WP4_CMA_001[3] | The AMASS tools must support specification of variability at the argumentation level | The system shall provide the capability for modelling arguments in the assurance case about multi-concern and multi-context. The multi-concern and multi-context argumentation could follow a variability modelling a solution. If GSN-like modelling elements are considered, the diamond for representing alternatives as well as the octagon for extrinsic variability could be considered. (1) | BVR Tool + OpenCert |
| WP4_CMA_002 | Component contracts must support multiple concerns | The system shall provide a contract specification language that supports the formalisation of both safety and security requirements. | CHESS |
| WP4_CMA_003 | Contract based multi-concern assurance | The system must support features that support contract based assurance with respect to multiple concerns; i.e. it must be possible to specify relations between safety contracts, security contracts and other-concerns-related contracts in order to take care of the influence of system modifications for mitigating the risks associated with one quality attribute on the contract belonging to another quality attribute. | OpenCert |
| WP4_SDCA_001 | System dependability co-architecturing and co-design | The system shall provide features, which allow architecture modelling collaboration and co-designing a system or component with a balanced combination of different goals addressing various quality attributes. | ANP tool, Concerto-FLA |
| WP4_SDCA_002 | System dependability co-verification and co-validation | The system shall support efficient system or component co-verification and co-validation with respect to multiple quality attributes. (2) | WEFACT, Medini Analyzer, Safety Architect, CHESS, FMVEA |
| WP4_SDCA_003 | The system shall allow combinations of safety and security analysis | The system shall allow combinations of safety and security analysis. | WEFACT, Medini Analyzer, Safety Architect, ANP tool,FMVEA, Concerto-FLA, MORETO |
| WP3_APL_004 | Architectural Patterns suggestions | The system could provide the user suggestions about a certain safety/security mechanism stored as architectural patterns. | MORETO |
| WP3_SC_005 | Requirements allocation | The system must provide the capability for allocating requirements to parts of the component model. More in general, requirements traceability shall be enabled. | MORETO |
| WP3_VVA_006 | Automatic provision of HARA/TARA-artifacts | The system shall provide the capability for automating HARA (Hazard Analysis Risk Assessment)/TARA (Threat Assessment & Remediation Analysis)-related artefacts (e.g., FTA, FMEA, attack trees). | FMVEA, MORETO, MediniAnalyze, SafetyArchitect, CHESS |
| WP3_VVA_009 | Capability to connect to tools for test case generation based on assurance requirements specification of a component/system | The system shall be able to connect to external tools to execute the test cases already specified. | WEFACT |
| WP5_CW_004 | Collaborative re-certification needs & consequences analysis | The AMASS Tool Platform shall support the collaboration among assurance managers and assurance engineers for re-certification needs & consequences analysis. | WEFACT, OpenCert |
| WP5_CW_005 | Collaborative system V&V | The AMASS Tool Platform shall support the collaboration among systems engineers for system V&V. | WEFACT |

[3] This requirement is shared between WP4 and WP6.

| Requirement No | Name | Description | Tool |
|---|---|---|---|
| WP5_CW_007 | Collaborative assurance evidence management | The AMASS Tool Platform shall support the collaboration among assurance managers and systems engineers for assurance evidence management. (3) | WEFACT, OpenCert |
| WP5_EM_016 | Evidence report generation | The AMASS Tool Platform shall be able to automatically generate reports, checklists, and evidence for certification purposes. | WEFACT |
| WP5_CM_001 | Modelling of standards | The AMASS tools shall be able to model a set of industrial standards (including the parts, objectives, practices, goals/requirements, criticality levels from the standards) | MORETO, WEFACT |
| WP5_CM_002 | Tailoring of Standards models to specific projects | The AMASS tools shall enable the tailoring of Standards models to specific project (e.g., by establishing the parts of the Standard that apply to a given assurance project). | WEFACT |
| WP6_CM_008 | Process Compliance (informal) management | The AMASS tools shall enable users to visualize process compliance. This means showing the links between the requirements and the applicant's evidence (during the planning as well as execution phase).<br>This visualization could be done via compliance maps (matrix) or via arguments aimed at justifying the satisfaction of the requirements coming from the standards. (3) | WEFACT, OpenCert, EPF-C |
| WP6_PPA_003 | Semi-automatic generation of process arguments | The system should be able to semi-automatic generate fragments of an assurance case for process arguments based on the process followed to develop a component/system. | WEFACT |

(1) Functionality mainly described in D6.2 [28].
(2) WEFACT allows combining V&V activities (e.g. calls to test tools) in one complex activity.
(3) Partially implemented.

Column "Requirement No" refers to the IDs in the deliverable D2.1 [16].

Each tool together with the implementation that implements requirements is shortly outlined in the following tool specific sections.

### 2.2.1 Requirements implemented in the Assurance Case Editor in OpenCert (*)

The Assurance Case Editor is part of the OpenCert project. It includes one of the basic building blocks for AMASS, the Assurance Case specification block. In this iteration, we have extended it in order to cover more of the requirements elicited for WP4 and solve some of the problems identified during the validation of previous prototype Core. Some of the requirements are covered partially and planned to be improved in future iterations.

Requirements implemented in the Assurance Case Editor in OpenCert are included in Table 2.

**Table 2.** Requirements implemented in the Assurance Case Editor

| Requirement No | Name | Description |
|---|---|---|
| WP4_ACS_001 | Assurance case edition | The system shall be able to edit an assurance case in a scalable way. |
| WP4_ACS_002 | Argumentation architecture | The system shall be able to edit a modular structure (argument architecture) associated with a system and/or component. |
| WP4_ACS_003 | Drag and drop argumentation patterns | The system shall be able to instantiate in the actual assurance case an argument pattern (concerning safety and security) selected from the list of patterns stored. |

| | | |
|---|---|---|
| **WP4_ACS_005** | Provide a structured language to the text inside the claims | The system could be able to provide support for language formalisation inside argument claims. |
| **WP4_ACS_007** | Argumentation import/export | The system could be able to import/export argumentations to SACM. |
| **WP4_ACS_010** | Composition of the overall argument | The system should provide the capability of generating a compositional assurance case argument. |
| **WP4_DAM_001** | Capability to model relationships between concerns | The system shall be able to provide an assurance case which records the relationships between dependability attributes and how they are affected because of design decisions. |
| **WP4_DAM_002** | Capability to capture conflicts occurring during system development and the trade-off process | The system shall provide the capability for modelling a dependability case, which captures the conflicts that occur during system development, explicitly show the dependencies of a design decision in relation with other assertions. |

Some of the requirements were implemented in the core prototype and lately improved in the second iteration. In the third iteration (P2), no further changes were needed, implementation has focused in resolving bugs.

**WP4_ACS_001: Assurance case edition**

This requirement was previously covered in Prototype Core.

**WP4_ACS_002: Argumentation architecture**

This requirement is focused on functionality “Assurance case structure navigation”, which was already implemented in Prototype Core. Assurance Case editor lets the user include argument modules in the diagram. This concept permits to encapsulate arguments (claims, strategies and evidences inside them). To see the encapsulated arguments, the user just needs to double click on the argument module and a tab with the argument diagram containing the arguments will be opened. All the elements inside the argument module are included in the model. The idea is to make feasible to apply modular argumentation concepts. We are able to encapsulate arguments of the same kind in argument modules. The way of classification might differ depending on the user. The user might want to encapsulate process arguments in an argument module, product arguments in another argument module and confidence arguments in another argument module, or rather to align the argumentation with the different components from the different suppliers that form the system and the adequacy of its integration.

**WP4_ACS_003: Drag and drop argumentation patterns**

This requirement was implemented in Prototype Core. However, one of the feedback comments received mentioned that the argument patterns needs to be stored locally as files before. With the new improvement the argument patterns can be stored either locally as files, or stored in a common repository. The user has a view where (s)he can browse the folders including patterns, select one, drag from the “templates” view and drop it in the actual diagram. The editor will copy the elements in the model and the position of the elements in the diagrams in a transparent way to the user.

**WP4_ACS_005: Provide a structured language to the text inside the claims**

This requirement was already covered in Prototype Core. There have not been any improvements regarding this requirement as there was no feedback from the case studies.

**WP4_ACS_007  Argumentation import/export**

This requirement has been covered briefly in the second iteration (Prototype P1). The user could provide a file storing an argument model specified using SACM to the actual argument model. Similarly, an argument model created in the Assurance Case editor can be exported to a file.

**WP4_ACS_010: Composition of the overall argument**

This requirement was partially covered in previous prototype (Prototype Core) and improved in prototype P1. In Prototype Core, in the argumentation diagram, the user could explicitly include the argument contract figure to show that there is a rationale behind the composition of the linked argument modules. An argument contract should be linked with at least two or more argument modules. With the new improvements in P1 the arguments that show the rationale for the connection are connected. A new argument diagram is associated with the contract figure and can be shown and edited when double clicking in the contract figure.

**WP4_DAM_001: Capability to model relationships between concerns**

This requirement has been covered in the second iteration (Prototype P1). In deliverable D4.2 [24] the "dependency relationship" has been presented. The new implementations tried to cover this new dependency relationship concept. No further development was needed in prototype P2.

### 2.2.2 Requirements realized in EPF-Composer & BVR Tool (*)

As it was recalled in D4.3 [25], EPF Composer is the tool that implements the EPF (Eclipse Process Framework) [9] approach for supporting customizable (software) process engineering frameworks.

In AMASS, the EPF approach and its tool support have been integrated as core building block. Within WP6, D6.2 [28] and D6.3 [47], EPF-C has been strengthened via integration with the BVR tool [3],[4]. This integration is beneficial not only for general reuse but more specifically for co-assessment and cross-concern reuse, focusing on the interplay of safety and security in line with WP4 objectives (see requirement WP4_CMA_001). This integration permits a user to model SiSoPLs (Security-informed Safety-oriented Process Lines). During the co-assessment, safety and security engineers are in the position to identify and systematize the overlapping region (commonality) and the variations.

An initial exploration of co-assessment and cross-concern reuse is documented in D6.2 [28] and D6.3 [47] . D4.7 [27], instead, includes in-depth guidance on how to benefit from such integration in the context of multi-concern (co-) assessment. Additional guidelines are expected to be provided in the final version D4.8 [57].

EPF Composer has also be strengthened with respect to compliance management. In the context of WP6, functionalities for generation of process-based arguments as well as compliance checking have been designed and implemented. These functionalities are relevant also in the context of WP4 (see requirement: WP6_PPA_003) since they have the potential of enabling the generation of co-assessment-related arguments as well as co-assessment proofs.

### 2.2.3 Requirements implemented in CHESS (*)

#### 2.2.3.1 Modelling different concerns for system components (*)

Different concerns/properties for system components can be represented in the architecture model by using the CHESS modelling language (CHESSML [35]) and then analysed (WP4_SDCA_001 requirement). In particular, (a subset of) MARTE [36] is available in CHESSML to allow modelling of timing concerns. Moreover, a dependability profile has been incorporated in CHESSML to allow modelling of safety properties (e.g. fault, error, failure and failure propagation); see Section 3.7.

In the context of AMASS, the extension of CHESS [15] to cover the modelling and analysis of security aspects has been investigated, in particular by considering what is already available from other modelling tools (e.g. Safety Architect provided by ALL4TEC), trying to understand if specific integration at modelling language and/or tool can be realized. Moreover, CHESSML has been extended to cover the modelling of security aspects and co-analysis is supported via the extension of ConcertoFLA (see Section 3.7).

The concept of component contract, the latter also available in CHESSML, can also be used to model properties of different concerns (WP4_CMA_002, WP4_CMA_003 requirements). Contracts can be derived according to obtained analysis results; for instance, a safety contract about failure propagation between input and output ports of a given component could be derived from CHESS by executing failure propagation analysis, the latter enabled by the failures-related information stored in the model by using the CHESS dependability profile. In the same manner, performance contract about worst-case response time of a component's operation could be derived after worst-case response time analysis performed in CHESS by using the timing MARTE annotations. Contracts can also be created as formalisation of system components requirements by using dedicated languages, for instance the temporal logic ones currently proposed in WP3.

To better represent the concern addressed by a given contract, CHESSML has been extended to support the notion of concern (e.g. safety, security, performance) attached to component contract. It is worth noting that the concern tag could also be derived automatically from the requirement(s) which is(are) formalised by given contract, assuming that the requirement comes with such information too. The assurance engineer can then use the information of concern attached to contracts to have a better understanding of the dependencies between concerns along the system architecture. For instance, he/she could reason about the relationships modelled for contracts, e.g. contracts refinement, to argue if a contract of a given concern depends on (is decomposed by in case of contracts refinement) contracts related to other concerns. CHESS has been extended also to compare the results of analysis applied in different architectures to for contract-based trade-off analysis (WP4_CAC_010). This comparison is enhanced by the parametrization of the architecture in which different architectures correspond to different configuration / assignments to the parameters. Each parameter can be a symbolic representation of a design choice. Contract-based trade-off analysis provides a characterization of which design choices affect the fulfilment of system and component contracts.

Additional guidelines including illustrative figures are expected to be provided in the final version D4.8 [57].

#### 2.2.3.2 Additional CHESS Functionalities (*)

In addition to the features for modelling different concerns for system components, CHESS was used for further features supporting modelling dependability aspects and semi-automatic generation of product arguments. For these developments, no implementation work was needed anymore in iteration 3. Nevertheless, they were elaborated at least conceptually and documented in D4.3 [25] In the following, a short description of these features is given.

**Modelling dependability aspects (**)**

As it was documented in D3.3 [23], CHESS implements the conceptual metamodel called SafeConcert [29]. SafeConcert enables dependability architects to model dependability's information necessary to conduct dependability analysis. SafeConcert is a subset of CHESSML (which in turn is an extension of SySML [30]), the meta-model used in CHESS toolset to enable component-based systems design. ConcertoFLA [31] allows users (system architects and dependability engineers) to decorate component-based architectural models (specified using CHESSML) with dependability-related information, execute Failure Logic Analysis (FLA) techniques, and get the results back-propagated onto the original model. Both SafeConcert and ConcertoFLA have been extended to support the modelling and analysis of security aspects (see Section 3.7 for details).

**Semi-automatic generation of product arguments (**)**

The Argument Generator plugin is implemented in CHESS. It generates a set of argument-fragments from the selected CHESS model and stores them in the corresponding destination assurance case in the CDO repository stated in the OpenCert preferences. Components in the CHESS model are decorated with contracts that are primarily used to verify that the model satisfies a particular requirement. The contract check is performed in OCRA from CHESS. To assure that the requirement is satisfied with sufficient confidence, we need to assure confidence in the contracts as well. Hence, we provided support in CHESS for enriching the contracts with assurance information. Argument Generator uses that information and creates an argument-fragment for each component and its related contracts. To support multi-concern assurance, we have extended the contracts and requirements specification in CHESS with a concern attribute to indicate that the particular contract/requirement is related to the selected concern. Based on this information, we generate argument-fragments that are concern-specific by filtering the component elements based on the concern tag. Currently, we indicated the concern in the name of the argument-fragment file. However, we are searching for a way to capture the concerns in the argumentation metamodel. The attached screenshots (Figure 1- Figure 6) illustrate the usage of the Argument Generator plugin. Further improvements of the generation are under way.

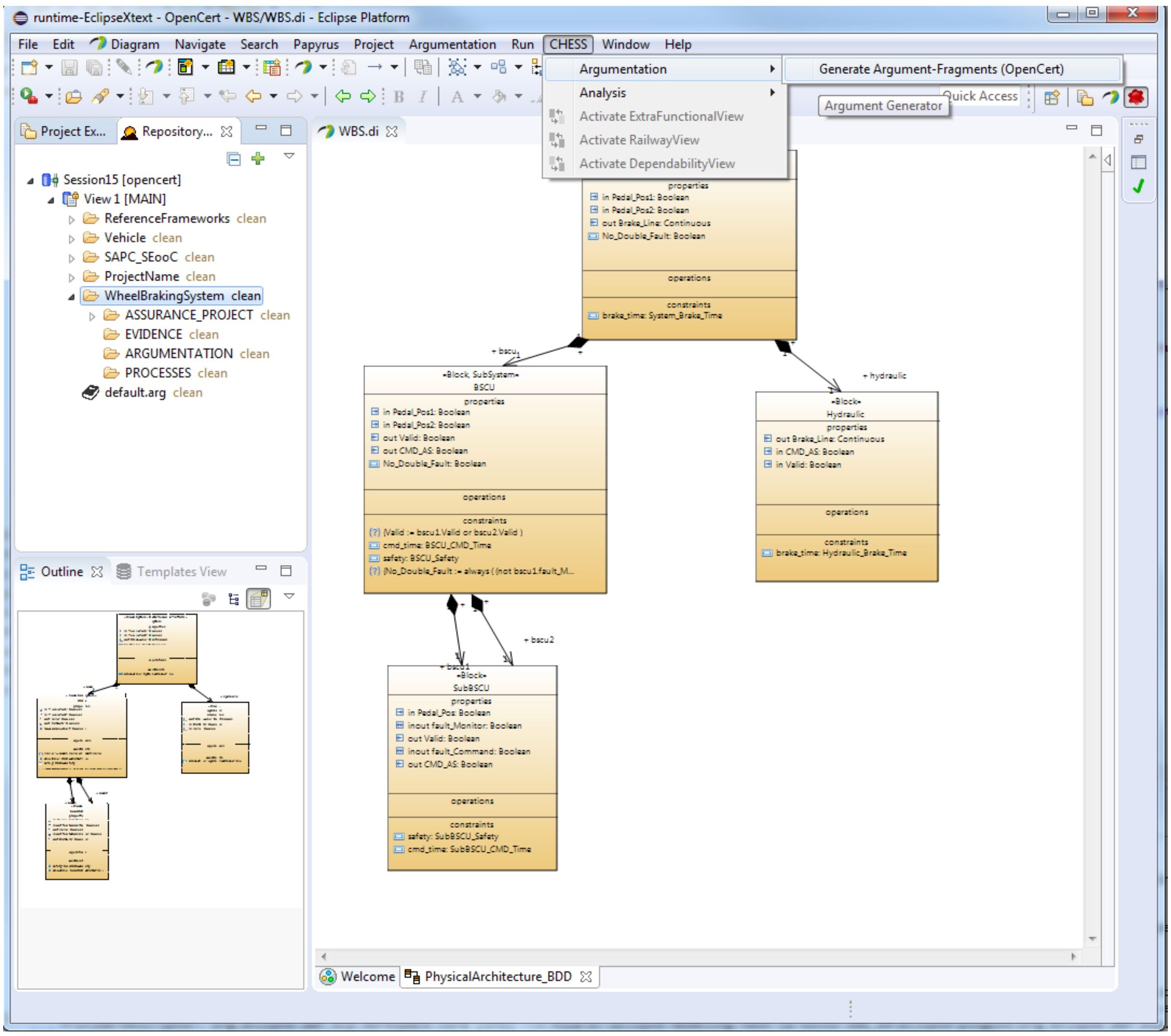
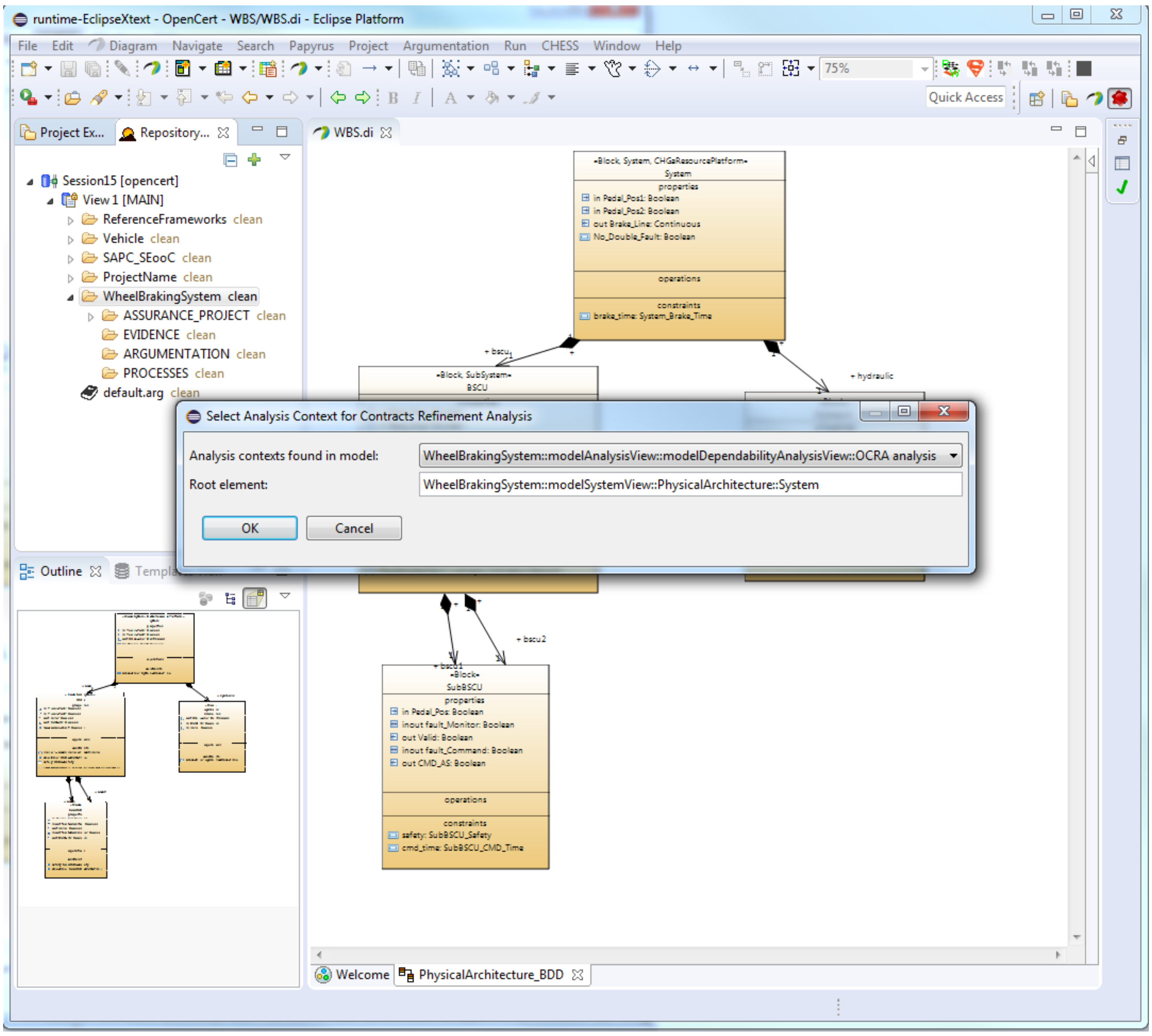


**Figure 2.** Initiating the argument-fragment generation (Step 1)


runtime-EclipseXtext - OpenCert - WBS/WBS.di - Eclipse Platform
File Edit Diagram Navigate Search Papyrus Project Argumentation Run CHESS Window Help
Quick Access
Project Ex...
Repository...
WBS.di
Outline
Welcome
PhysicalArchitecture_BDD

Session15 [opencert]
View 1 [MAIN]
ReferenceFrameworks clean
Vehicle clean
SAPC_SEooC clean
ProjectName clean
WheelBrakingSystem clean
ASSURANCE_PROJECT clean
EVIDENCE clean
ARGUMENTATION clean
PROCESSES clean
default.arg clean

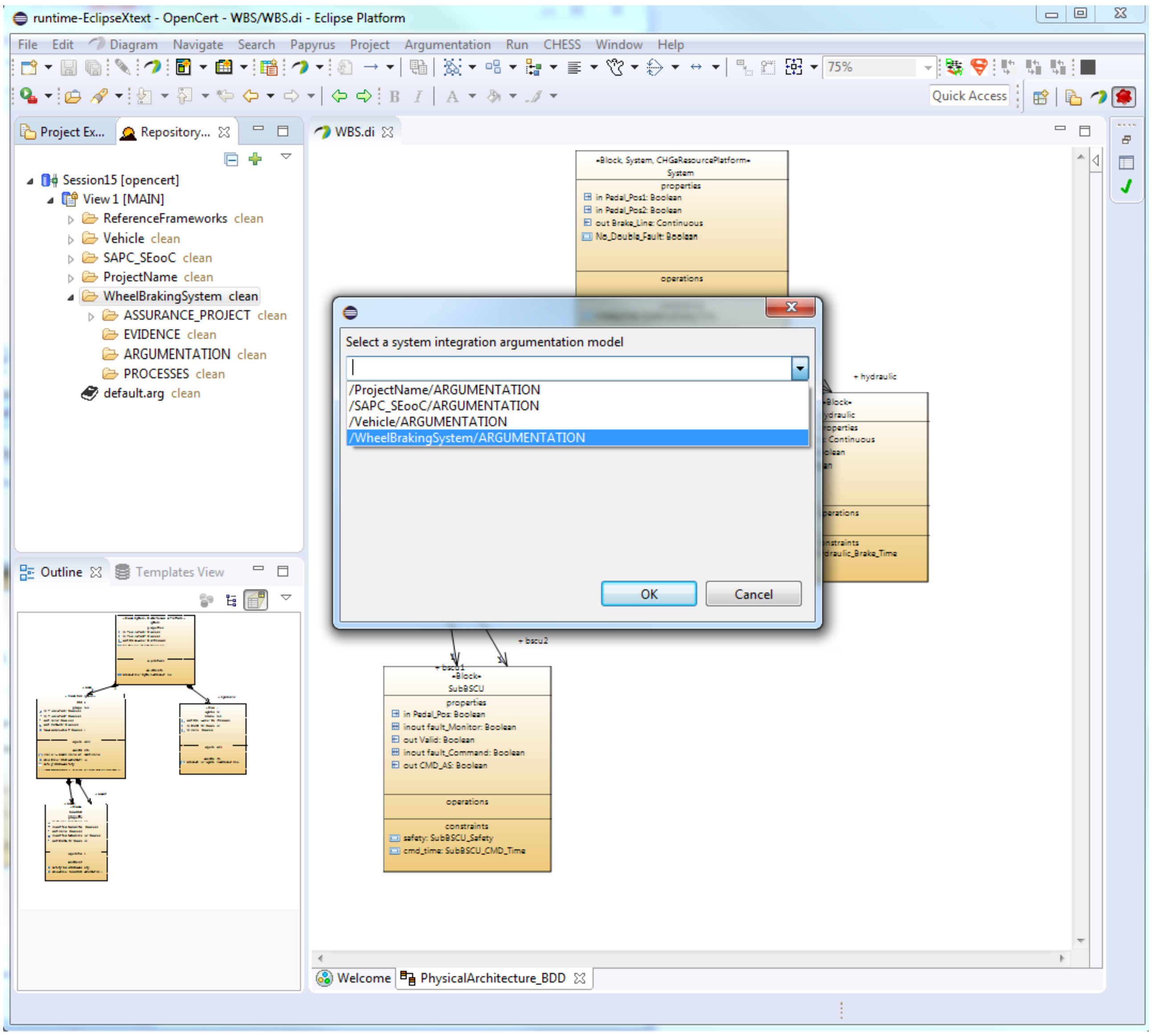
Select Analysis Context for Contracts Refinement Analysis

Analysis contexts found in model: WheelBrakingSystem::modelAnalysisView::modelDependabilityAnalysisView::OCRA analysis

Root element: WheelBrakingSystem::modelSystemView::PhysicalArchitecture::System

OK Cancel

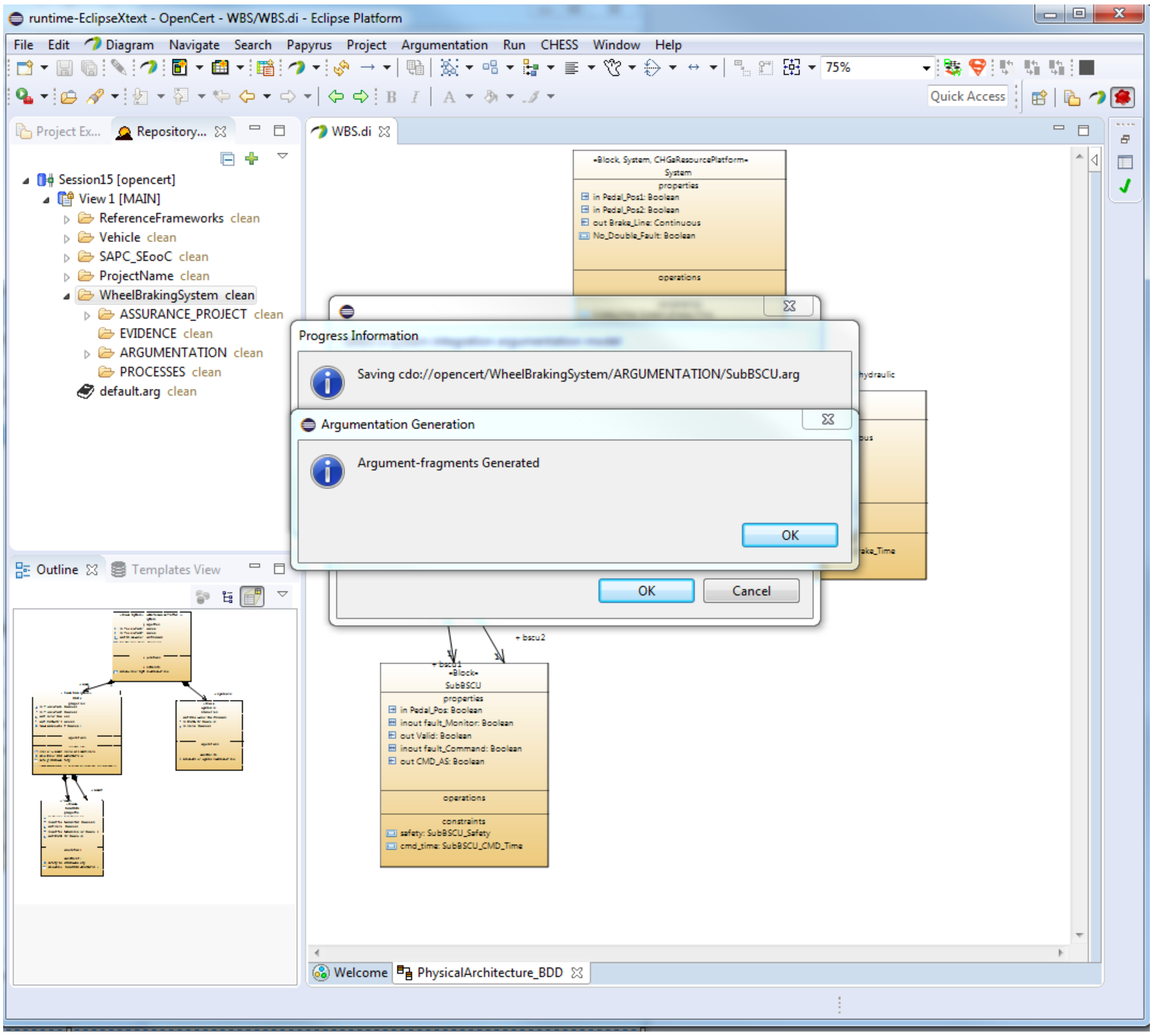
**Figure 3.** Selecting the source analysis context (Step 2)


runtime-EclipseXtext - OpenCert - WBS/WBS.di - Eclipse Platform
File Edit Diagram Navigate Search Papyrus Project Argumentation Run CHESS Window Help
Project Ex... Repository...
Session15 [opencert]
View 1 [MAIN]
ReferenceFrameworks clean
Vehicle clean
SAPC_SEooC clean
ProjectName clean
WheelBrakingSystem clean
ASSURANCE_PROJECT clean
EVIDENCE clean
ARGUMENTATION clean
PROCESSES clean
default.arg clean
Outline Templates View
WBS.di
Welcome PhysicalArchitecture_BDD
OK
Cancel

Select a system integration argumentation model

/ProjectName/ARGUMENTATION
/SAPC_SEooC/ARGUMENTATION
/Vehicle/ARGUMENTATION
/WheelBrakingSystem/ARGUMENTATION

**Figure 4.** Selecting the destination assurance case folder on the CDO repository (Step 3)



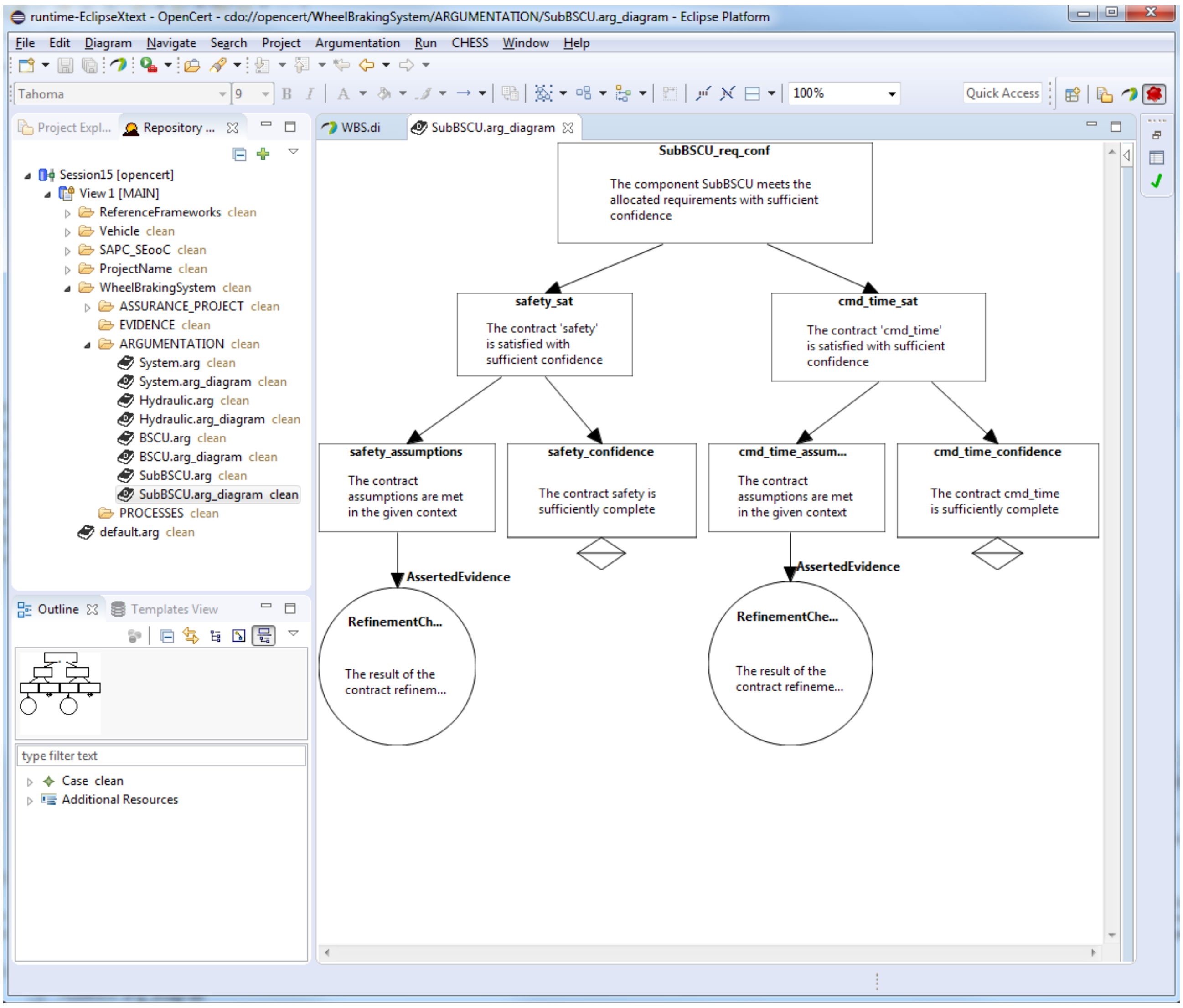
Progress Information

Saving cdo://opencert/WheelBrakingSystem/ARGUMENTATION/SubBSCU.arg

Argumentation Generation

Argument-fragments Generated

**Figure 5.** Generation successfully completed with argument-fragments for each block



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

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

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

**safety_assumptions**
The contract assumptions are met in the given context

**safety_confidence**
The contract safety is sufficiently complete

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

**cmd_time_confidence**
The contract cmd_time is sufficiently complete

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

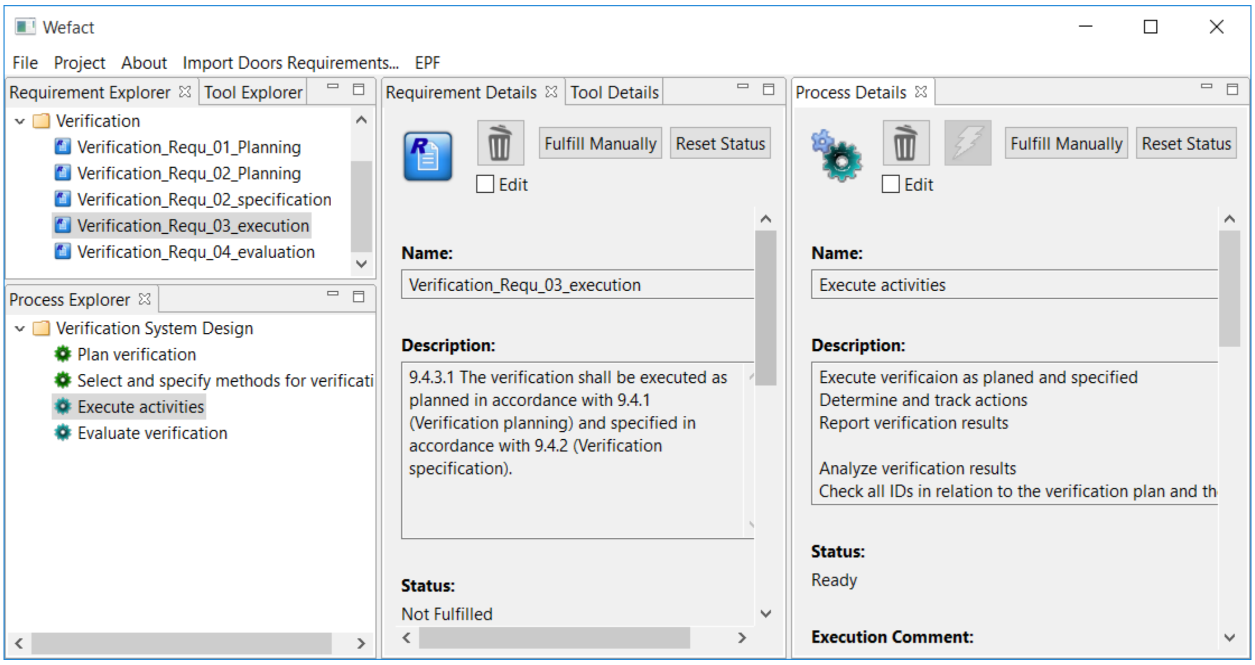
**RefinementChe...**
The result of the contract refineme...

**Figure 6.** An example of the generated argument-fragment

## 2.2.4 Requirements implemented in WEFACT (*)

WEFACT is an external tool for assurance workflow execution. It can use a process model defined in EPF-C or use process activities defined in WEFACT itself. In WEFACT, the activities of the EPF model are associated with V&V activities and respective tools, and WEFACT eventually executes these activities, keeping track of changes of associated artefacts (e.g. software modules under test) and the associated requirements. In this way, WEFACT supports continuous impact management in the event of changing requirements, models or implementations and triggers then only those re-assurance activities which are necessary as a consequence of the changes.

Figure 7 shows the WEFACT user interface after importing a process model, which appears in the "Process Explorer" in the lower left corner. In the middle the selected requirements is displayed, to the right the associated verification process and its status can be seen. More details can be found in D4.3 [25].


Wefact
File Project About Import Doors Requirements... EPF
Requirement Explorer
Tool Explorer
Verification
Verification_Requ_01_Planning
Verification_Requ_02_Planning
Verification_Requ_02_specification
Verification_Requ_03_execution
Verification_Requ_04_evaluation
Process Explorer
Verification System Design
Plan verification
Select and specify methods for verificati
Execute activities
Evaluate verification
Requirement Details
Tool Details
Fulfill Manually
Reset Status
Edit
Process Details

Name: Verification_Requ_03_execution

Description: 9.4.3.1 The verification shall be executed as planned in accordance with 9.4.1 (Verification planning) and specified in accordance with 9.4.2 (Verification specification).

Status: Not Fulfilled

Name: Execute activities

Description: Execute verificaion as planed and specified
Determine and track actions
Report verification results

Analyze verification results
Check all IDs in relation to the verification plan and th

Status: Ready

Execution Comment:

**Figure 7.** WEFACT user interface example after importing a process model

In the third iteration P2, WEFACT is integrated with the AMASS platform and fulfils the following requirements:

**Table 3.** Requirements [partly] implemented in WEFACT

| Requirement No | Name | Description |
|---|---|---|
| **WP4_ACS_006** | Provide guidelines for argumentation | The system could be able to provide guidelines about the assurance case edition based on the system/component development phase status. |
| **WP4_ACS_008** (1) | Traceability of the dependability case | The system should provide the dependability case reviewers the ability of tracing an overall dependability case (GSN) goal to the requirement within the dependability profile for a given system element and the attribute of interest with which goal is associated. |
| **WP4_SDCA_002** (1) | System dependability co-verification and co-validation | The system shall support efficient system or component co-verification and co-validation with respect to multiple quality attributes. |
| **WP4_SDCA_003** (1) | The system shall allow combinations of safety and security analysis | The system shall allow combinations of safety and security analysis. (2) |
| **WP3_VVA_009** | Capability to connect to tools for test case generation based on assurance requirements specification of a component/system | The system shall be able to connect to external tools to execute the test cases already specified. (3) |
| **WP5_CW_004** | Collaborative re-certification needs & consequences analysis | The AMASS Tool Platform shall support the collaboration among assurance managers and assurance engineers for re-certification needs & consequences analysis. |
| **WP5_CW_005** | Collaborative system V&V | The AMASS Tool Platform shall support the collaboration among systems engineers for system V&V. |
| **WP5_CW_007** | Collaborative assurance evidence management | The AMASS Tool Platform shall support the collaboration among assurance managers and systems engineers for assurance evidence management. |
| **WP6_CM_008** | Process Compliance (informal) management | The AMASS tools shall enable users to visualize process compliance. This means showing the links between the requirements and the applicant's evidence (during the planning as well as execution phase).<br>This visualization could be done via compliance maps (matrix) |

| Requirement No | Name | Description |
|---|---|---|
| | | or via arguments aimed at justifying the satisfaction of the requirements coming from the standards. |
| WP5_EM_016 | Evidence report generation | The AMASS Tool Platform shall be able to automatically generate reports, checklists, and evidence for certification purposes. |
| WP6_CM_001 | Modelling of standards | The AMASS tools shall be able to model a set of industrial standards (including the parts, objectives, practices, goals/requirements, criticality levels from the standards) |
| WP6_CM_002 | Tailoring of Standards models to specific projects | The AMASS tools shall enable the tailoring of Standards models to specific project (e.g., by establishing the parts of the Standard that apply to a given assurance project). |
| WP6_PPA_003 | Semi-automatic generation of process arguments | The system should be able to semi-automatic generate fragments of an assurance case for process arguments based on the process followed to develop a component/system. |

1) Partly implemented, i.e. WEFACT achieves this by combining workflows.
2) WEFACT allows combined safety and security analyses by combining calls to separate safety and security analysis tools in one activity.
3) WEFACT supports calling tools in the executed assurance activities; this includes calls to test tools. In this sense, WEFACT can be used as a test automation engine. This feature, in fact, supports WP3 requirement WP3_VVA_009.

In the following, the implementation in WEFACT is shortly described for each of the above mentioned requirements.

**WP4_ACS_006 Provide guidelines for argumentation**

Together with EPF, WEFACT offers opportunities to guide the user through certain assurance activities at defined points in the workflow. These assurance activities can be any activity in the lifecycle like, for instance, safety analysis, performance analysis, software design, system test, reviews, validation activities, etc. This functionality has been implemented in prototype P1.

**WP4_ACS_008 Traceability of the dependability case**

The WEFACT workflow supports the recognition of evidences which are invalidated by modification of requirements or input artefacts of assurance activities.

**WP4_SDCA_002 System dependability co-verification and co-validation**

WEFACT can be instantiated as workflow engine for verification of any quality attribute. This is possible in conformance with a process model created with EPF-C or stand-alone with WEFACT. WEFACT can, as far as possible, automatically start tools for verifying or validating deliberate properties or quality attributes of the system or the artefact under consideration. The UMA process model says what shall be verified/validated, and WEFACT allows to couple this step to appropriate tool[s] and to execute the workflow.

**WP4_SDCA_003 The system shall allow combinations of safety and security analysis.**

WEFACT can support processes for controlling separate as well as combined safety and security analyses. In Iteration 2 WEFACT can be used to combine calls to separate safety and security analysis tools in a complex analysis step. In iteration 3, combined methods for co-analysis are expected (FMVEA, Medini Analyzer).

**WP3_VVA_009 Capability to connect to tools for test case generation based on assurance requirements specification of a component/system**

WEFACT offers various bindings for tools, among others, test case generation tools. WEFACT maintains a list of tools including their bindings; the user can associate assurance steps (process activities) with tools.

WEFACT allows interdependent sequences of tool calls so that, as an example, a successful call to a test case generation tool can be linked to a subsequent call to a test tool executing the generated test cases.

**WP5_CW_004 Collaborative re-certification needs & consequences analysis**

WEFACT allows multiple users to use its database and provides - based on its continuous impact management w.r.t. changes of requirements and system artefacts - support for efficient, resource-saving re-certification.

**WP5_CW_005 Collaborative system V&V**

WEFACT supports collaborative, workflow-controlled V&V, integrated with the assurance case.

**WP5_CW_007 Collaborative assurance evidence management**

While and after gathering assurance evidences, WEFACT supports assurance managers and systems engineers in tracking the progress of the evidence collection for finalizing the assurance case.

**WP6_CM_008 Process Compliance (informal) management**

WEFACT shows the dynamic status of compliance with Standards as the imported EPF process model is inherently standards-compliant. While designing as well as executing the assurance workflow, WEFACT shows at any point in time the actual fulfilment of the product requirements as well as the process requirements from the standards.

**WP5_EM_016 Evidence report generation**

WEFACT creates evidences for requirements and can provide reports (currently in a textual version) about their fulfilment status. This can be used for the certification documentation.

**WP6_CM_001 Modelling of standards**

WEFACT allows modelling the process steps for creating the work products in compliance with standards.

**WP6_CM_002 Tailoring of Standards models to specific projects**

WEFACT allows to use predefined process models (e.g. an EPF model for a specific standard) and to tailor it to the domain-specific practices or to the individual project.

**WP6_PPA_003 Semi-automatic generation of process arguments**

As described in D6.3 [47], WEFACT can create Argument Fragments from modelled (typically standards-conformant) assurance processes.

### 2.2.5 Requirements implemented in the FMVEA tool (**)

In the third iteration, FMVEA is integrated with the AMASS platform as external tool and fulfils the following requirements:

**Table 4.** Requirements implemented in the FMVEA tool

| Requirement No | Name | Description |
|---|---|---|
| WP4_SDCA_002 | System dependability co-verification and co-validation | The system shall support efficient system or component co-verification and co-validation with respect to multiple quality attributes. |
| WP4_SDCA_003 | The system shall allow combinations of safety and security analysis | The system shall allow combinations of safety and security analysis. |
| WP3_VVA_006 | Automatic provision of HARA/TARA-artifacts | The system shall provide the capability for automating HARA (Hazard Analysis Risk Assessment)/TARA (Threat |

| | | |
|---|---|---|
| | | Assessment & Remediation Analysis)-related artefacts (e.g., FTA, FMEA, attack trees). |

In the following, the implementation in FMVEA is shortly described for each of the above-mentioned requirements.

**WP4_SDCA_003 The system shall allow combinations of safety and security analysis.**
FMVEA allows combined multi-concern analyses, based on a system model, currently focused on safety and security. Extensions towards views on more quality attributes are planned for future research projects.

**WP4_SDCA_002 The system shall support efficient system or component co-verification and co-validation with respect to multiple quality attributes.**
After implementing the necessary (e.g. architectural) mitigation measures, definition of possible threats and failure modes the parameterized (e.g. define the focus points inside the modelled system) analysis can be executed. Components inside the model can be selected individually, this possibility saves costs in time.

**WP3_VVA_006 Automatic provision of HARA/TARA-artifacts**
FMVEA provides automated hazard and threat analyses with risk assessment using its failure and threat database.

### 2.2.6 Requirements implemented in the ANP tool (**)

In the third iteration, a prototype of the ANP (Analytic Network Process) tool has been provided. Its integration with the AMASS platform is currently (August 2018) manual, but work is ongoing to integrate with the AMASS platform using the CHESS dependability profile. The ANP tool supports System analysis as well as Trade-off analysis and fulfils the requirements listed in Table 5.

The system shall allow combinations of safety and security analysis.

**Table 5.** Requirements implemented in the ANP tool

| Requirement No | Name | Description |
|---|---|---|
| WP4_CAC_0101) | Contract-based trade-off analysis | The system could provide the capability to evaluate safety and security requirements on different system architectures to perform trade-off analysis based on the contract specification. |
| WP4_DAM_002 | Capability to capture conflicts occurring during system development and the trade-off process | The system shall provide the capability for modelling a dependability case which captures the conflicts that occur during system development and the trade-off process to justify why the taken design decisions are the most optimal ones. |
| WP4_SDCA_001 | System dependability co-architecturing and co-design | The system shall provide features, which allow architecture modelling collaboration and co-designing a system or component with a balanced combination of different goals addressing various quality attributes. |
| WP4_SDCA_003 | The system shall allow combinations of safety and security analysis | The system shall allow combinations of safety and security analysis. |

1) Implemented in a specific way, see explanation below.

In the following, the implementation in the ANP tool is shortly explained for each of the above-mentioned requirements.

**WP4_CAC_010 Contract-based trade-off analysis**

This requirement is supported in a specific way. Using patterns in the ANP tool implies a design contract between the assumption that an architectural or design solution is implemented and the guaranteed improvement of the achieved level of the addressed quality attributes like for instance reliability, availability, attack or failure detection likelihood or effort needed for cracking a key.

**WP4_DAM_002 Capability to capture conflicts occurring during system development and the trade-off process**

Assuming different security patterns, the ANP approach allows to determine the impact on safety. Based on attack likelihood, failure rates and the impact, the user can choose a solution between different variants based on a numeric risk value.

**WP4_SDCA_001 System dependability co-architecturing and co-design**

The ANP tool uses security patterns together with field experience data to determine a quantitative attack likelihood measure and provides safety and security risk numbers to support choosing optimal architectural solutions.

**WP4_SDCA_003 The system shall allow combinations of safety and security analysis**

The quality attributes availability and reliability are related to implementations for safety and security. The ANP tool provides availability and reliability analysis via Time NET and can also be extended for timing analysis, which addresses an aspect of performance analysis. Summarizing, the ANP is able to support even a wider range of multi-concern analyses than the requirement demands.

### 2.2.7 Requirements implemented in the Concerto-FLA extension (**)

In the third iteration, the Concerto-FLA tool is integrated with the AMASS platform as external tool and fulfils the following requirements.

**Table 6.** Requirements implemented in the Concerto-FLA tool

| Requirement No | Name | Description |
|---|---|---|
| WP4_SDCA_001 | System dependability co-architecturing and co-design | The system shall provide features, which allow architecture modelling collaboration and co-designing a system or component with a balanced combination of different goals addressing various quality attributes. |
| WP4_SDCA_003 | The system shall allow combinations of safety and security analysis | The system shall allow combinations of safety and security analysis. |

In the following, the implementation in the Concerto-FLA tool is shortly described for each of the above-mentioned requirements.

**WP4_SDCA_001 System dependability co-architecturing and co-design.**

The extension of the dependability profile of CHESSML [52]discussed in Section3.7.1 enables the modelling of security in addition to the available support of modelling of safety properties as mentioned in Section 2.2.3.1.

**WP4_SDCA_003 The system shall allow combinations of safety and security analysis.**

The extension of Concerto-FLA enables co-analysis (safety and security) and the generation of FTs from the analysis results. This extension is explained in detail in Section 3.7.1.

## 2.2.8 Requirements implemented in MORETO (**)

The Enterprise Architect plugin MORETO (Model-based Security Requirements Management Tool) was used in Case Study 1 and is therefore already described in D1.5 [46]. It was decided to include it in the third iteration P2 of the AMASS platform as an external tool. The tool is now integrated with the AMASS platform and fulfils the following requirements (see Table 7).

**Table 7.** Requirements implemented in the MORETO tool

| Requirement No | Name | Description |
|---|---|---|
| WP4_SDCA_003[1)] | The system shall allow combinations of safety and security analysis | The system shall allow combinations of safety and security analysis. |
| WP3_VVA_006[1)] | Automatic provision of HARA/TARA-artifacts | The system shall provide the capability for automating HARA (Hazard Analysis Risk Assessment)/TARA (Threat Assessment & Remediation Analysis)-related artefacts (e.g., FTA, FMEA, attack trees). |
| WP3_SC_005 | Requirements allocation | The system must provide the capability for allocating requirements to parts of the component model. More in general, requirements traceability shall be enabled. |
| WP6_CM_001 | Modelling of standards | The AMASS tools shall be able to model a set of industrial standards (including the parts, objectives, practices, goals/requirements, criticality levels from the standards) |
| WP3_APL_004 | Architectural Patterns suggestions | The system could provide the user suggestions about a certain safety/security mechanism stored as architectural patterns. |

1) Partly implemented – restricted to the security part.

In the following, the implementation in the MORETO tool is shortly described for each of the above-mentioned requirements.

**WP4_SDCA_003 The system shall allow combinations of safety and security analysis**

MORETO allows security analyses, which covers only part of WP4_SDCA_003. By WEFACT, however, MORETO can be combined with a safety analysis tool resulting in a combined analysis.

**WP3_VVA_006 Automatic provision of HARA/TARA-artefacts**

After modelling the system architecture including certain security-relevant parameters, MORETO uses its knowledge to automatically generate security requirements and a set of standards-conformant security controls as mitigation measures. The use of the tool is, however, restricted to security.

**WP3_SC_005 Requirements allocation**

MORETO supports – in conformance with selected cybersecurity standards – the automatic allocation of cybersecurity requirements to model elements, e.g. components.

**WP6_CM_001 Modelling of standards**

Up to now (August 2018), two cybersecurity standards, namely for IEC 62443 [48] and IEEE 1686 [49] are supported by MORETO.

**WP3_APL_004 Architectural Patterns suggestions**

In automatic mode, MORETO creates security controls which represent patterns to cope with security threats. More details about such pattern can be found in D3.6 [39]. In manual mode, it is also possible to define own threat mitigation measures.

### 2.2.9 Requirements implemented in Medini Analyzer (**)

In the third iteration, the *Medini Analyzer* tool (for safety and cybersecurity) is used as an external tool to the AMASS platform and fulfils the following requirements (see Table 8).

**Table 8.** Requirements implemented in the *Medini Analyzer* tool

| Requirement No | Name | Description |
|---|---|---|
| WP4_SDCA_003 | The system shall allow combinations of safety and security analysis | The system shall allow combinations of safety and security analysis. |
| WP3_VVA_006 | Automatic provision of HARA/TARA-artifacts | The system shall provide the capability for automating HARA (Hazard Analysis Risk Assessment)/TARA (Threat Assessment & Remediation Analysis)-related artefacts (e.g., FTA, FMEA, attack trees). |
| WP3_VVA_010 | Model-based safety analysis | The system shall allow the user to generate fault trees and FMEA tables from the behavioral model and the fault injection. |
| WP3_SC_005 | Requirements allocation | The system must provide the capability for allocating requirements to parts of the component model. More in general, requirements traceability shall be enabled. |

In the following, the implementation in the *Medini Analyzer* tool is shortly described for each of the above-mentioned requirements.

**WP4_SDCA_003 The system shall allow combinations of safety and security analysis**

Medini supports both, safety and cyber-security analysis at the same time or separated from each other. Both analysis can be based on the same system (SysML) model in combination. Safety and security artefacts can have relationship or dependencies among each other (e.g. safety and security requirements, vulnerabilities and failures, assets and system elements).

**WP3_VVA_006 Automatic provision of HARA/TARA-artefacts**

Medini has rich built-in capabilities to automate certain tasks in the overall tool. That automation capabilities are covering all aspects of the tool, including system modelling but also derivation of initial safety and security analysis stubs. These capabilities can be used to write new automations, but there are already a few standard automations included. Example automation already included in the derivation of fault trees from Simulink or SysML models, the derivation of TARA skeletons from Attack Trees and the derivation of FMEA/FMEDA form sheets from system models, to name a few.

**WP3_VVA_010 Model-based safety analysis**

Medini analysis methods (security and safety) are solely based on system models. Analysis artefacts as fault trees, HARA or TARA or FMEA worksheets refer to system model or failure model elements and reflect element attributes as for example failure rates in FTA event probabilities.

**WP3_SC_005 Requirements allocation**

Medini supports rich traceability with different levels of semantics, from pre-defined “hard” relationships as “Allocation” or “Contribution” relationships to generic and customizable “Trace” relations. Requirements can be freely allocated to system elements, also to larger sets of elements as for example required for semiconductors. The allocation can be utilized by automation features that for example propagate Integrity Levels (as the ASIL) owned by a requirement down to the system architecture (system model).

## 2.3 Installation and User Manuals (*)

It is planned to provide an exhaustive installation description with the steps necessary to install the third prototype as well as a user manual for the **internal tools** in [20] together with D2.5 "AMASS user guidance and methodological framework" [45], which will not be repeated here. That document will contain all required steps and document references to set up the tools.

**External tools** have a stub description in the AMASS User Manual and possibly manuals on the tool provider website. The entire AMASS user manual will be published in D2.5 [45], which is due in m31 (October 2018). Table 9 depicts an overview on available installation documentation and user manuals for the external tools implemented in iterations 2 and 3.

**Table 9.** Available installation documentation and user manuals for external tools implemented in iteration 2 and 3

| Tool | Available Installation Documentation and User Manual |
|---|---|
| WEFACT | https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/WEFACT_UserManual.docx<br>https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/WEFACT_Installation_Guide.docx |
| FMVEA | https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/FMVEA_UserManual.docx<br>https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/ FMVEA _Installation_Guide.docx |
| ANP tool | https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/ANP-tool_UserManual.docx<br>https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/ANP-tool _Installation_Guide.docx |
| *Concerto-FLA extension* | https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/*Concerto-FLA* -tool_UserManual.docx<br>https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2*Concerto-FLA*-tool _Installation_Guide.docx |
| MORETO | https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/MORETO_UserManual.docx<br>https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/MORETO_Installation_Guide.docx |
| Medini Analyzer | https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/MediniAnalyzer_UserManual.docx<br>https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP2/MediniAnalyzer_Installation_Guide.docx |

The Assurance Case Editor, CHESS and EPF-Composer are part of the Core platform; their description is therefore contained in the general AMASS user manual. All other tools described in this deliverable are external. Their documentation is available at the tool providers; nevertheless, for ease of use, the documentation including User Manual and Installation Guide is provided to the AMASS project participants in an internal svn directory, as shown in Table 9.

The AMASS SVN code repository is open to AMASS partners with the same credentials as the SVN document repository. In case that people outside the project need access, please contact the AMASS Project Manager (alejandra.ruiz@tecnalia.com).

# 3. Implementation Description (*)

## 3.1 Assurance Case Editor from OpenCert

### 3.1.1 Description of Features Implemented in P1

In accordance with the deliverable D2.3 [18], the components that are part of the Assurance Case Manager Component have been implemented within the Assurance Case Editor from OpenCert and it covers the following blocks: the Assurance Case Management and partially the Contract-based Multi-concern Assurance, this second one just related to argument contracts.

The Assurance Case Management block is an Eclipse-Based Argumentation Editor. It contains plugins for editing argumentation models and plugins for management of argument patterns and module libraries. (Please note that the term "module" used for argumentation modules differs from the "implemented modules" described in this chapter.)

The *Assurance Case Editor* is responsible for the Argument model creation and edition. The purpose of the *Argument Patterns/Module Management* tool is to provide services storing and instantiating modular argumentation and patterns. The *Dependability modelling* tool is responsible for managing the "dependability relationship" described in D4.2 [24].

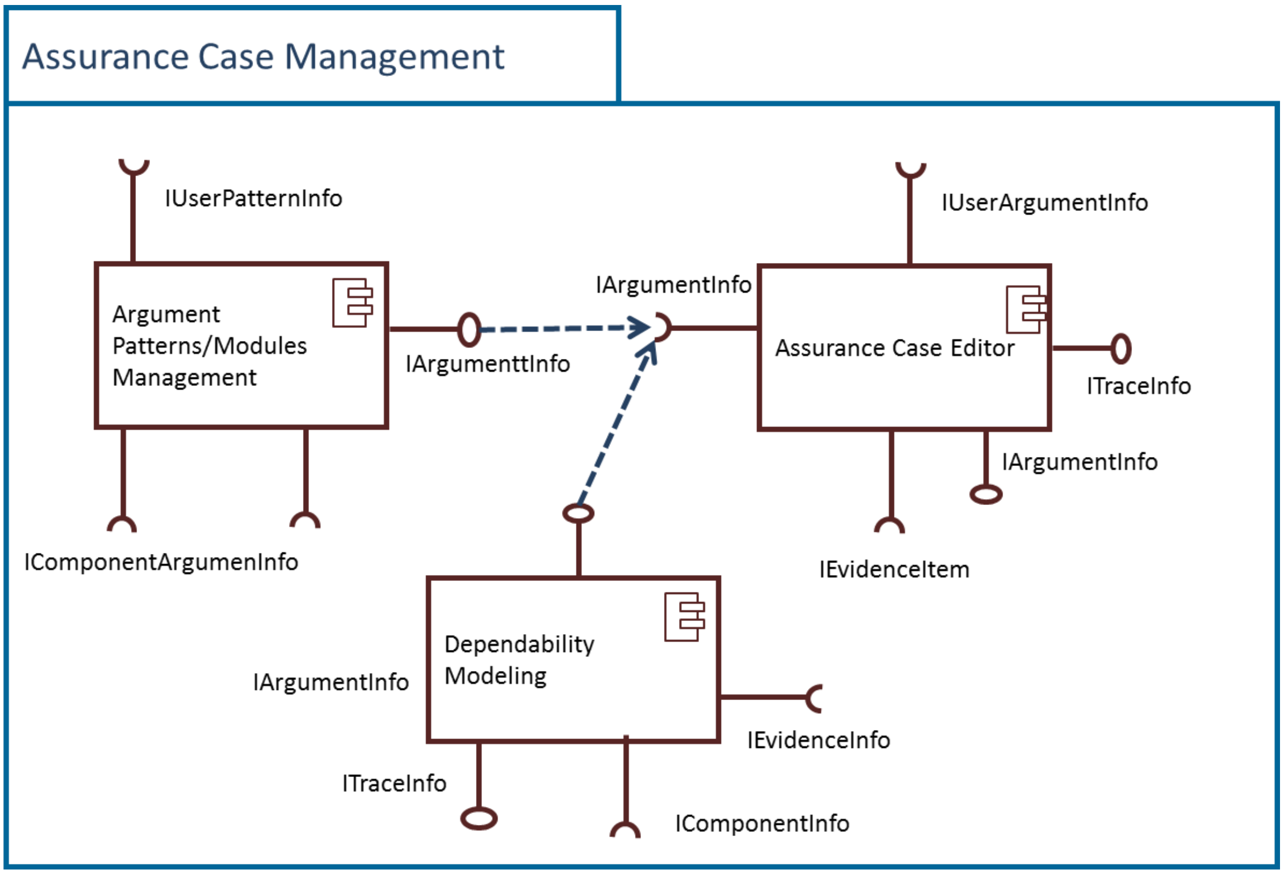


**Figure 8.** Tool modules for Assurance Case Management Component

Above that, the Assurance Case Editor, also covers partially the edition or the argument contracts using the contract-based multi-concern assurance module, from the Contract management component. It deals with argument contracts and it is highly connected with the modular argumentation services.

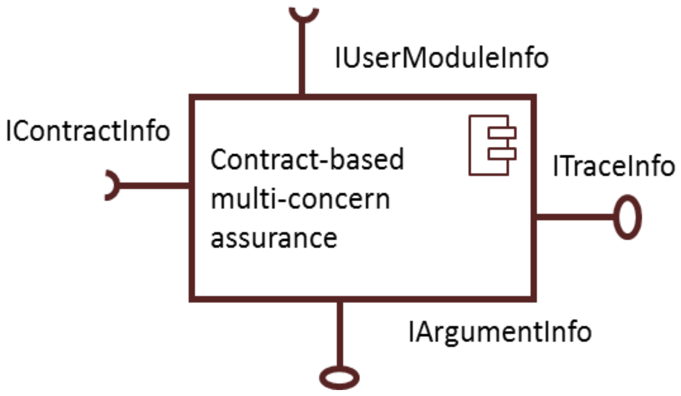


**Figure 9.** Tool module from Contract Management Component

In this second iteration of the AMASS platform, main work has been done to consolidate the results from the first iteration. The main problem has been the navigation associated with the modular argumentation and the migration of the argument modules and patterns from files to database storage. The contract-based multi-concern assurance block and the dependability modelling block, which appeared in Figure 8, have been implemented in this second iteration.

The technologies used to develop the Assurance Case Editor are:

- To generate Editors: GMF [7], EMF [8], Eugenia [12]
- For model transformations: Epsilon (ETL) [9]
- For storage: CDO [13]
- For vocabulary: Xtext [11]

### 3.1.2 Description of Features Implemented in P2 (**)

As mentioned before, no new features have been implemented in the third iteration. The implementation has focused in resolving bugs and consolidating previous developments.

### 3.1.3 Source Code Description

The source code of the AMASS prototype can be found in the source code SVN repository at [19].The code for the assurance case modules second prototype will be stored together with the other basic building blocks in the repository under “tag” to distinguish the state of the code at the time of the integrated release.

Once all the plugins are installed, these are the necessary ones for the Assurance Case Management and the Contract-based Multi-concern Assurance:

- GSN.figures
  This plugin provides utilities to draw model elements according to the Goal Structuring Notation (GSN) standard [6].
- org.opencert.sam.arg
  In this plugin, the argumentation metamodel is defined and stored, and the Java implementation classes for this model are generated.
- org.opencert.sam.arg.diagram
  This plugin is the diagram editor itself. It manages diagrams and includes a canvas to draw on, a palette with creation tools and default selecting and zooming capabilities, a property view and an outline view.
- org.opencert.sam.arg.edit
  The edit plugin includes adapters that provide a structured view and perform command-based edition of the model objects.

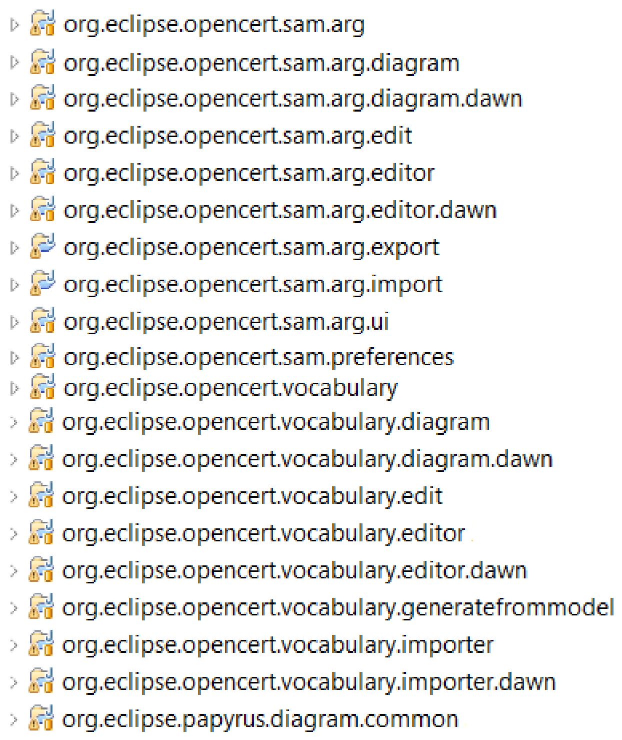
- org.opencert.sam.arg.editor
  This plugin provides the user interface to view instances of the model using several common viewers and to add, remove, cut, copy and paste model objects, or to modify the objects in a standard property sheet.
- org.opencert.sam.arg.export
  This plugin provides adapters to export an argument model stored in the common database, to an argument model specified using SACM in a file.
- org.opencert.sam.arg.import
  This plugin provides adapters to import an argument model specified using SACM in a file to an argument model to be stored in the common database.
- org.opencert.sam.arg.ui
  This is an additional plugin. It offers several utilities such as drawing model elements not included in the GSN standard, accessing to argument patterns and modules.
- org.opencert.sam.arg.preferences
  This plugin manages the default preferences required by the Argumentation diagram editor. The parameters which can be defined are the Modules Directory (with all argumentation modules stored from previous argumentation phases) and the Patterns Directory (that contains all argumentation patterns templates).
- org.opencert.sam.vocabulary
  Contains the vocabulary meta model, which is part of the previous results from OPENCOSS CCL (Common Certification Language).
- org.opencert.sam.vocabulary.edit
  The edit plugin includes adapters that provide a structured view and perform command-based edition of the model objects. It contains the CCL vocabulary meta model respective the related EMF based tree editor and GMF based graphical editor to create and edit vocabulary models.
- org.opencert.sam.vocabulary.editor
  This plugin provides the user graphical interface to view instances of the model using an EMF based tree editor and GMF based graphical editor to create and edit vocabulary models.

In addition, the following plugins are necessary to manage the assurance project and to handle the corresponding evidences:

- org.opencert.apm.assuranceassets
  In this plugin, the assurance assets metamodel is defined and stored, and the Java implementation classes for this model are generated.
- org.opencert.apm.assuranceassets.edit
  The edit plugin includes adapters that provide a structured view and perform command-based edition of the assurance assets model objects.
- org.opencert.evm.evidspes
  In this plugin, the evidence metamodel is defined and stored, and the Java implementation classes for this model are generated.
- org.opencert.evm.evidspec.edit
  The edit plugin includes adapters that provide a structured view and perform command-based edition of the model objects.
- org.opencert.infra.properties
  This plugin contains the definition of the Property metamodel, and the Java implementation classes for this model.
- org.opencert.infra.properties.edit

In relation with the edit plugin for evidence, this plugin contains a provider to display the model in a user interface.

Figure 10 illustrates the list of plugins described above.



```
org.eclipse.opencert.sam.arg
org.eclipse.opencert.sam.arg.diagram
org.eclipse.opencert.sam.arg.diagram.dawn
org.eclipse.opencert.sam.arg.edit
org.eclipse.opencert.sam.arg.editor
org.eclipse.opencert.sam.arg.editor.dawn
org.eclipse.opencert.sam.arg.export
org.eclipse.opencert.sam.arg.import
org.eclipse.opencert.sam.arg.ui
org.eclipse.opencert.sam.preferences
org.eclipse.opencert.vocabulary
org.eclipse.opencert.vocabulary.diagram
org.eclipse.opencert.vocabulary.diagram.dawn
org.eclipse.opencert.vocabulary.edit
org.eclipse.opencert.vocabulary.editor
org.eclipse.opencert.vocabulary.editor.dawn
org.eclipse.opencert.vocabulary.generatefrommodel
org.eclipse.opencert.vocabulary.importer
org.eclipse.opencert.vocabulary.importer.dawn
org.eclipse.papyrus.diagram.common
```

**Figure 10.** Assurance Case Specification plugins

## 3.2 EPF-Composer Tool

### 3.2.1 Description of Realized Features

As mentioned before, features aimed at strengthening EPF-C [9] with respect to compliance management as well as process-related variability management via the integration with the BVR Tool were designed and implemented in WP6. In AMASS WP4, these functionalities can be used for creating and tailoring the project-specific assurance workflow models starting from a standard-specific models, generating process-based arguments as well as proofs for co-assessment. Thus, in this deliverable, they are not recalled. The reader may refer to WP6-deliverables as well as related publications.

More specifically, in WP6, a specific plugin ("Seamless Integrator") has been conceived for interacting with the BVR tool. This extension is described in D6.3 [47] and was also accepted for publication at SPLC-2018 [54]. Concerning the generation of process-based arguments, a paper was accepted at QUATIC-2018 [55].

Finally, concerning the strengthening of EPF-C for compliance checking, a paper was accepted at SEAA-2018 [56].

### 3.2.2 Source Code / Interface Description

EPF Composer is a publicly available tool operating on the open UMA process metamodel format. It is part of the AMASS Core platform. Its porting was done in the context of AMASS WP6/WP7 and presented at EclipseCon-2018 [56].

## 3.3 CHESS Tool (*)

### 3.3.1 Description of Features Implemented in P1

As introduced in section 2.2.3.1, CHESS modelling language has been extended to allow the decoration of contract w.r.t. the concern addressed by the contract itself. In the CHESS profile for contract specification, the information about the concern is attached to the *FormalProperty* entity (Figure 11), the latter representing a (UML) constraint that can play the role of assumption or guarantee property of a given contract. In this way, it is possible for the user to decorate the contract with information related to the concern addressed by the contract itself (e.g. safety, security, performance).

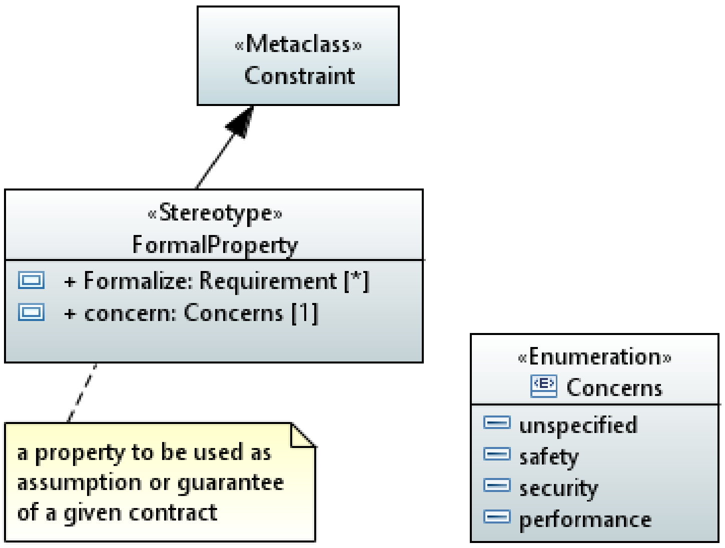

«Metaclass» Constraint
«Stereotype» FormalProperty
+ Formalize: Requirement [*]
+ concern: Concerns [1]
a property to be used as assumption or guarantee of a given contract
«Enumeration» Concerns
unspecified
safety
security
performance

**Figure 11.** Contract profile supporting modelling of concerns

CHESS functionality has been extended with an Argument Generator plugin that utilises the assurance and concern specific information attached to the contracts and facilitates generation of argumentation fragments for each component in the system model. As the different contracts and related assurance information are concern specific, the Argument Generator builds concern-specific argument-fragments.

### 3.3.2 Description of Features Implemented in P2 (**)

**Contract-based trade-off analysis in parameterized architectures**

A parametrized architecture is an architecture in which the number of components, the number of ports, the connections, and the static attributes of components depends on a (possibly infinite) set of parameters.

The trade-off analysis implemented in P2 requires the instantiation of the parameterized architecture.

This activity takes as input the parameterized architecture and the configurations provided by the user, i.e. the assignments of the set of parameters. For each instance of the architecture that is derived from the configurations, one or more contract-based analyses are performed. Such analyses are described in the D3.6 [45].

The trade-off analysis is the process that compares the contract-based analyses results and generates a report. Figure 12 shows the input/output artefacts used to perform a trade-off analysis.

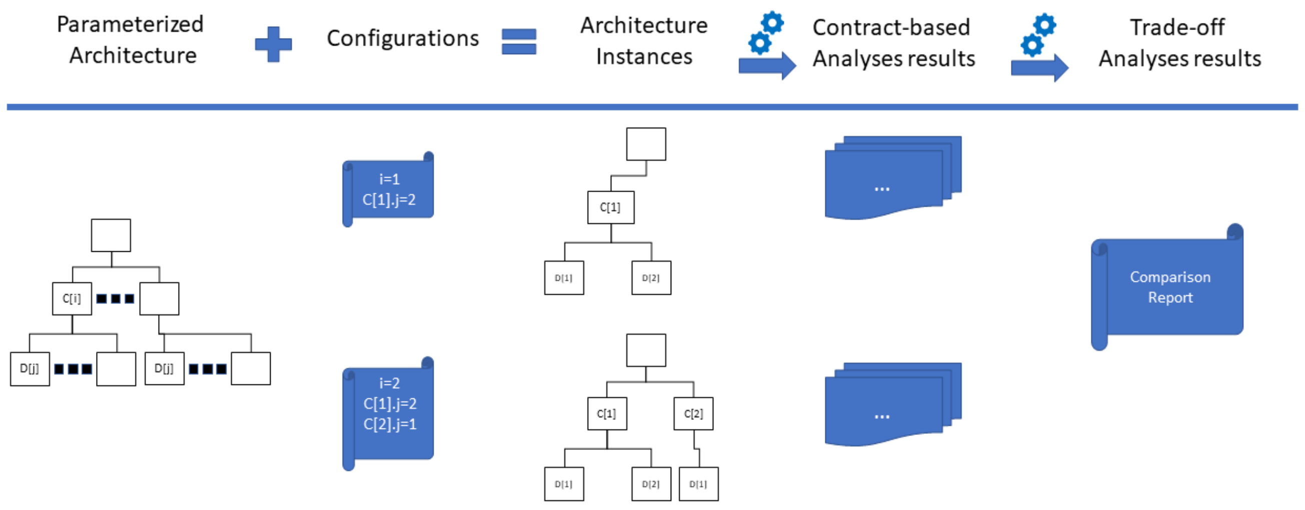


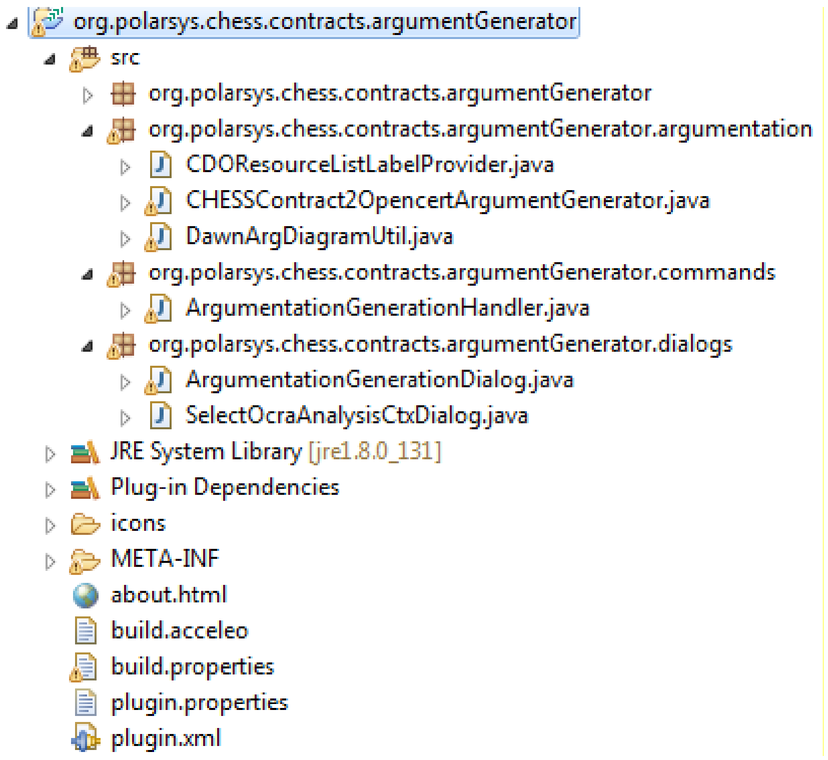
**Figure 12.** The parameterized architecture and the configurations are used to generate the architecure instances. For each instance, a set of contract-based analyses is performed. Their results will be compared and integrated in a report format

### 3.3.3 Source Code Description (*)

The CHESS modelling language extension presented in the previous section has been implemented in Eclipse by extending the CHESS UML profile for contract specification, in particular by using the support available in Papyrus for what regards the modelling of UML profiles. Then Java code representing the profile implementation has been automatically (re)generated starting from the UML profile definition by using Eclipse EMF[4] facilities. The obtained Java code has been embedded in a dedicated plugin to allow the usage of the CHESS profile for contract specification while modelling with the Papyrus/CHESS editor.

The Argument Generator functionality presented in the previous section has been implemented as an eclipse plugin. The source code structure is presented in Figure 13. The plugin first prompts the selection of the OCRA analysis context used as the source of the CHESS system model for which refinement analysis has been performed. Then, Argumentation generation dialog is started to select the destination for the generated argument-fragments. CHESSContract2OpencertArgumentGenerator.java performs the information extraction from the CHESS model and argumentation creation in the selected assurance case on the CDO repository.

[4] https://www.eclipse.org/modeling/emf/



```
◢ org.polarsys.chess.contracts.argumentGenerator
  ◢ src
    ▷ org.polarsys.chess.contracts.argumentGenerator
    ◢ org.polarsys.chess.contracts.argumentGenerator.argumentation
      ▷ CDOResourceListLabelProvider.java
      ▷ CHESSContract2OpencertArgumentGenerator.java
      ▷ DawnArgDiagramUtil.java
    ◢ org.polarsys.chess.contracts.argumentGenerator.commands
      ▷ ArgumentationGenerationHandler.java
    ◢ org.polarsys.chess.contracts.argumentGenerator.dialogs
      ▷ ArgumentationGenerationDialog.java
      ▷ SelectOcraAnalysisCtxDialog.java
  ▷ JRE System Library [jre1.8.0_131]
  ▷ Plug-in Dependencies
  ▷ icons
  ▷ META-INF
    about.html
    build.acceleo
    build.properties
    plugin.properties
    plugin.xml
```

**Figure 13.** Argument Generator plugin source

At the current stage (August 2018), the implementation of the trade-off analyses is in progress. However, we plan to include this feature in the final release of the P2.

## 3.4 WEFACT Tool (*)

### 3.4.1 Description of Features Implemented in P1

**WEFACT for Multi-concern Activities**

WEFACT implements a workflow for assurance activities of various kinds, like analyses, design activities, testing, verification, and many others. These assurance process activities can be safety-oriented, security-oriented or performance-oriented, and they can as well address any other quality attribute. WEFACT allows to deliberately combine such quality-attribute-oriented activities in parallel or in sequential order. This allows, even in the absence of combined multi-concern-engineering tools, a defined structure of co-engineering processes. As an example, WEFACT can be configured to combine a safety analysis-oriented HARA tool with a security-oriented TARA tool, thus implementing safety-security-co-analysis with separate tools.

#### 3.4.1.1 Structure of WEFACT

The goal of WEFACT is to support the complete 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.

**Requirement:**

Requirements are defined as the entities needed to achieve the objectives of the project. 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 Requirements **enter the state of completion.** A Requirement can hold a connection to predefined processes. If all processes are executed successfully, the Requirement's status changes to "fulfilled".

**Process:**
Processes describe the steps that need to be conducted. The principal for the structure of Processes conforms to the structure of the Requirements mentioned earlier. Top-level Processes consist of sublevel Processes and the top-level Process reaches the status Successful once all sub-processes have been executed without errors. Each Process can be linked to one or more Requirements. Moreover, a Workflow Tool can be associated to a certain Process. This way the Process becomes an executable which uses existing input and produces new output. This output can serve as input for subsequent Processes.

**Workflow Tool:**
A Workflow Tool represents an application or component that can be addressed via URL. By defining Workflow Tools inside WEFACT, these applications and components can be directly invoked. Solely type for the Workflow Tool, the path to the corresponding executable and some input arguments need to be specified.

#### 3.4.1.2 Integrated process execution

One of the main features of WEFACT is the option to execute processes directly from the application. Workflow Tools can be linked to multiple Processes in the workflow. Through this connection a process becomes equivalent to an executable.

WEFACT supports different types of process execution, **manual and automatic**. While manual tools require the user to save the results to a specific location, automatic tools return the results that are consequently evaluated and stored. **The outputs of the executed processes are stored in a centralized SVN.**

After the evaluation of the Process Result, the status of the executed Process and associated Requirements is modified.

Figure 14 shows the WEFACT Activity Diagram.

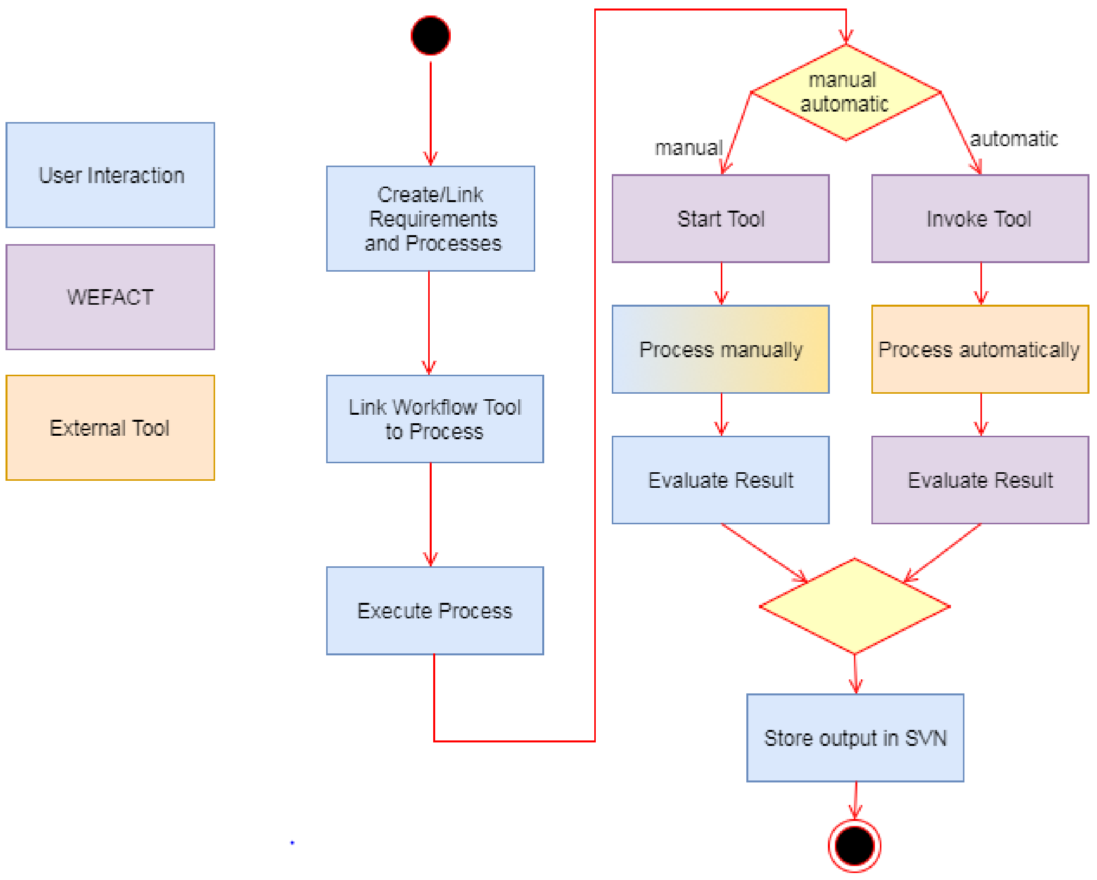


**Figure 14.** WEFACT Activity Diagram

#### 3.4.1.3 Centralized SVN Storage

The biggest advantage of storing artefacts on an SVN is the fact that every participant is granted access to the created evidence **by remote access/remotely**. Project partners work on the same corporate set of artefacts that allows all partners to work collectively on a common solution rather than on independent ones.

#### 3.4.1.4 EPF-C Model Import

A common approach to create a process-based workflow is the utilization of the Eclipse Process Framework Composer EPF-C [9]. EPF-C allows the user to specify a custom workflow and additional artifacts that are integrated into the workflow. These workflows can be exported as XML file.

WEFACT is capable of importing these XML files and translates the provided content into WEFACT Process Structure. Afterwards the imported workflow can be displayed in WEFACT, the created Processes may be linked and carried out.

### 3.4.2 Description of Features Implemented in P2 (**)

As a new capability, a feature for generating process-based argument fragments out of the process model is being developed in WEFACT. From a conceptual perspective, this functionality belongs to WP6 and is therefore described in D6.3 [47]. The implementation and the interface specification will be contained in D6.6 [50], which is due in m31, i.e. October 2018.

### 3.4.3 Interface Module Description (*)

WEFACT is an Eclipse application developed under the Eclipse-RCP. WEFACT implements an assurance workflow based on a project specific process model previously instantiated in EPF-C. The resulting UMA-compliant format is part of the CACM and EPF-C is part of the AMASS core platform.

WEFACT imports this UMA model and derives the WEFACT-specific execution model from EPF-C. Ex-post modifications of the originally imported process model within WEFACT are possible but it is recommended that these changes remain minimal. A Re-import of the changes in WEFACT into the UMA process model created with EPC-C is currently not foreseen.

WEFACT can treat process requirements (coming e.g. from a functional safety standard) as well as product requirements (functional and non-functional requirements related to user requirements as well as safety and security requirements to the product as defined during the HARA/TARA lifecycle phase). In order to enable WEFACT to control the entire assurance workflow, WEFACT must get all aforementioned requirements in order to operate on the full set of requirements. It has to be mentioned that usually not all requirements treated in WEFACT assurance activities are necessarily directly referenced in assurance case arguments; especially test cases will rather be referenced in a test result document, which is cited in a verification and validation report. The appropriate preparation of this verification/validation report is a process requirement, whose proven and appropriate preparation provides the evidence for a respective process argument instance. As a consequence, not all WEFACT results put into evidence model instances need to be linked to assurance case solutions.

The following Figure 15 shows an example for the relation between WEFACT results and the CACM in case all V&V activities are referenced as GSN solutions in the Argumentation trees of the Assurance Case.

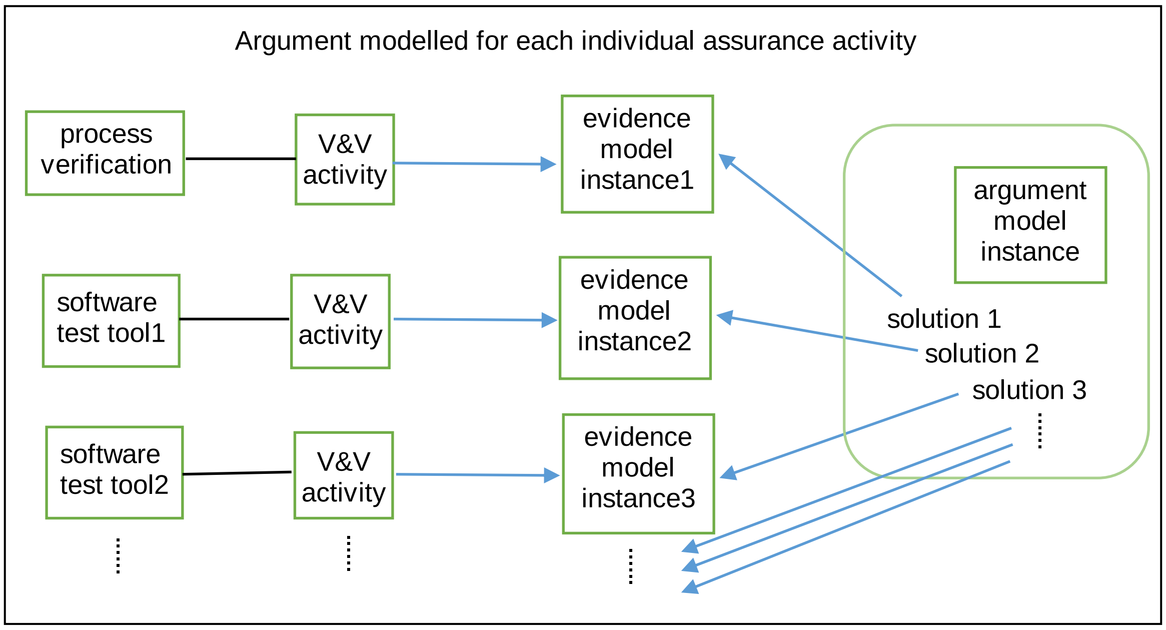


**Figure 15.** Relation between WEFACT low level activity results and the CACM

An Alternative is to have all evidence model instances linked to solutions from SACM arguments; in this case, individual V&V activities must be grouped if only one argument solution applies to them. Figure 16 shows graphically an example for the respective relation.

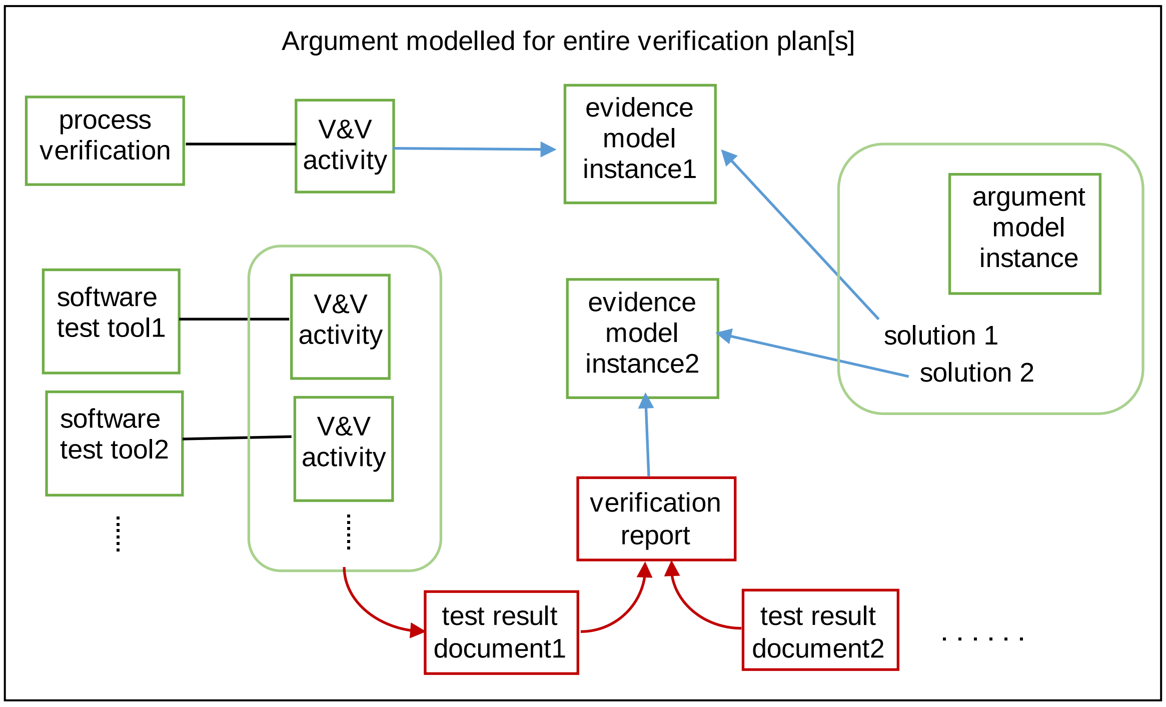


**Figure 16.** Relation modelled between WEFACT activity results on high level and the CACM evidences

More deeply staged process structures can be devised and are also possible from WEFACT side. EPF-C, however, supports only a maximum of three layers (package=phase, task, and step), thus limiting the applicability of such approaches in EPF-C context.

In AMASS, the argumentation for the assurance case can be manually created by means of the OpenCert Assurance Case Editor, which operates on the project specific instance of the Structured Assurance Case

Metamodel SACM, which is linked with the GSN solutions for these arguments in the respective evidence model instance.

WEFACT provides evidences per requirement, so each evidence in the SACM instance must be traceable to a requirement. WEFACT supports also the creation of requirements, WEFACT is further able to import requirements from a DOORS 9.6 database and, in a future version, to import ReqIF data and requirements from the XML file created by Papyrus from an UML Requirements Table.

A standard reference (RefStandards) pointing to a clause in e.g. a Functional Safety Standard is not provided in WEFACT, an implementation in the EPF-based UMA process model is basically possible in future versions.

Figure 17 shows the WEFACT Metamodel with exception of the links, which are explained separately.

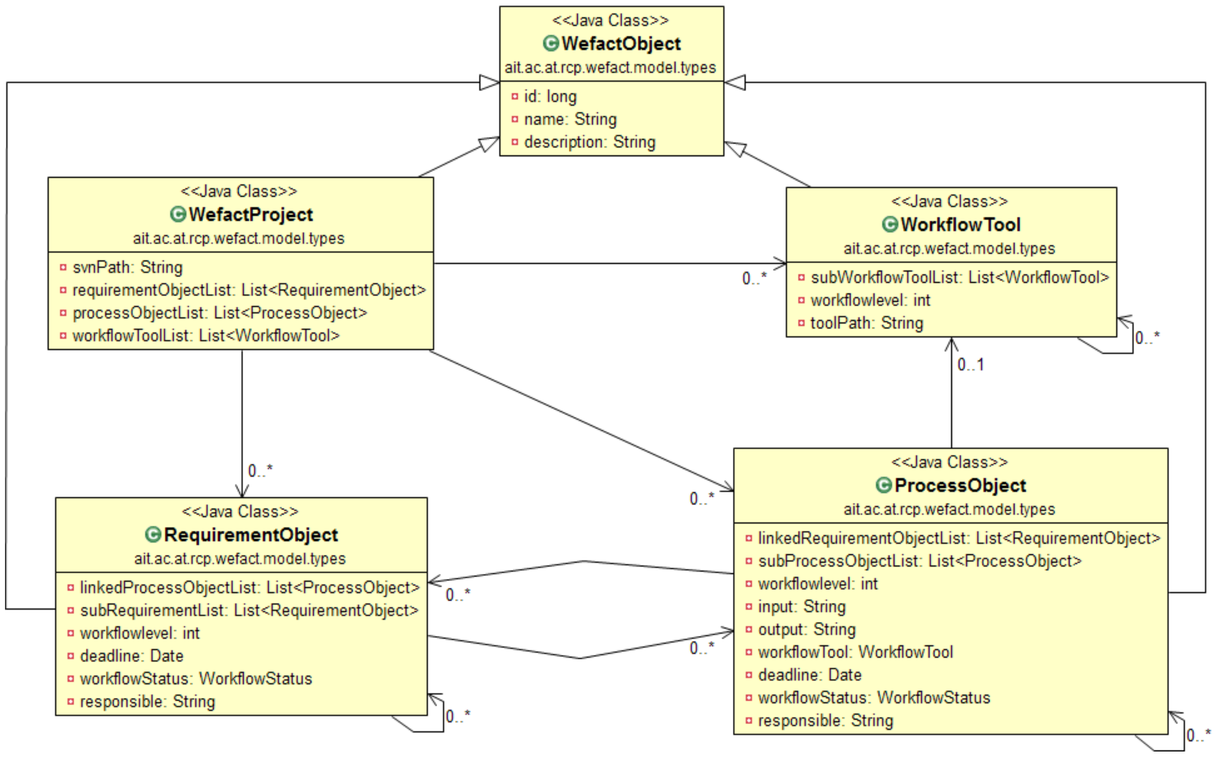

<<Java Class>>
WefactObject
ait.ac.at.rcp.wefact.model.types
id: long
name: String
description: String
<<Java Class>>
WefactProject
ait.ac.at.rcp.wefact.model.types
svnPath: String
requirementObjectList: List<RequirementObject>
processObjectList: List<ProcessObject>
workflowToolList: List<WorkflowTool>
<<Java Class>>
WorkflowTool
ait.ac.at.rcp.wefact.model.types
subWorkflowToolList: List<WorkflowTool>
workflowlevel: int
toolPath: String
0..*
0..*
0..1
<<Java Class>>
RequirementObject
ait.ac.at.rcp.wefact.model.types
linkedProcessObjectList: List<ProcessObject>
subRequirementList: List<RequirementObject>
workflowlevel: int
deadline: Date
workflowStatus: WorkflowStatus
responsible: String
<<Java Class>>
ProcessObject
ait.ac.at.rcp.wefact.model.types
linkedRequirementObjectList: List<RequirementObject>
subProcessObjectList: List<ProcessObject>
workflowlevel: int
input: String
output: String
workflowTool: WorkflowTool
deadline: Date
workflowStatus: WorkflowStatus
responsible: String
0..*
0..*
0..*
0..*
0..*
0..*

**Figure 17.** The WEFACT Metamodel

In the following, syntax and semantics of the classes and attributes are explained in detail.

**WefactObject**

ait.ac.at.rcp.wefact.model.types

- id: long
- name: String
- description: String

A WefactObject represents the WEFACT base class and need not be reflected in the assurance model instance. All WEFACT classes are derived from it.

**WefactProject**

ait.ac.at.rcp.wefact.model.types

- svnPath: String
- requirementObjectList: List<RequirementObject>
- processObjectList: List<ProcessObject>
- workflowToolList: List<WorkflowTool>

In terms of AMASS, a WefactProject represents an assurance project and comprises all project specific artefacts and model instances relevant to WEFACT. It is associated with

- a path in the svn (svnPath), where the artefacts of the project are stored,
- the list of requirements (requirementObjectList), for which V&V activities are provided in WEFACT,
- the V&V activities [to be] processed in WEFACT (processObjectList), and
- the tools associated to V&V activities and called by WEFACT (workflowToolList).

The **WefactProject** mapping to the CACM is depicted in the following table:

**Table 10.** WefactProject mapping

| **WEFACT** | **CACM** | | comment |
|---|---|---|---|
| element | Model | element | |
| svnPath | n/a | n/a | This is the SVN base path where the project repository is located. There is no directly corresponding element in CACM. The concrete SVN (etc.) locations of artefacts are individually given in the associated Resource location of the ManagedArtefact instances. |
| requirementObjectList | a) process requiremt.&toplevel prod.req.:UMA metamodel<br>b) basic product req.: Component MM | task, work product<br>Requirement | The requirements associated with the assurance project. The list contains toplevel as well as sub requirements, process and product requirements.<br>Details are explained in the section "RequirementObject" further below. |
| processObjectList | UMA metamodel | task, work product | The V-Plans and the V&V activities within the assurance project. |
| workflowToolList | SACM Artefact Metamodel | Technique | |

**RequirementObject**

ait.ac.at.rcp.wefact.model.types

- linkedProcessObjectList: List<ProcessObject>
- subRequirementList: List<RequirementObject>
- workflowlevel: int
- deadline: Date
- workflowStatus: WorkflowStatus
- responsible: String

**Product requirements** for WEFACT can come from different sources, e.g. the CACM. I particular, they can be:

- imported from a **DOORS database**,
- imported from a **ReqIF** file (in ARTA iteration 3),
- imported from a .xls file exported from the **Papyrus UML Requirements** table,
- imported from and exported into the **CACM Component Metamodel**,

and finally, they can be

- created in **WEFACT**.

**Process requirements** for WEFACT are usually:

- imported from the project-specific instantiation of the process model (created with EPF-C), which corresponds to the CACM **Process Metamodel** instance.

and, like product requirements, they can also be

- created in **WEFACT**.

Requirements in WEFACT can be nested, i.e. a top-level requirement can be subdivided into sub-requirements, which can be subdivided again and so forth. The evidence for the fulfilment of a requirement which has sub-requirements is composed by WEFACT from the fulfilment of these sub-requirements.

The **RequirementObject** mapping to the CACM is depicted in the following table:

**Table 11.** RequirementObject mapping

| WEFACT | CACM | | comment |
|---|---|---|---|
| element | Model | element | |
| linkedProcessObjectList | SACM Artefact Metamodel | Artefact, Activity | Artefacts and Activities can be linked together. The same principle applies to WEFACT Requirements and Processes. |
| subRequirementList | Component Metamodel | | The Component Metamodel does not contain a concept for nested requirements. So, as for product requirements, only lowest level WEFACT requirements can be mapped to. |
| workflowlevel | | | Internal only usage in WEFACT. |
| deadline | | | No representation in CACM. (1) |
| workflowStatus | ExecutedProcess Metamodel | Executed activity | WEFACT e.g. fulfilled, not fulfilled.<br>The semantics is that an instance of ExecutedActivity is created by WEFACT if the WEFACT activity has been successfully completed. If the respective WEFACT V-Plan or V&V activity becomes invalid by changes in the corresponding requirement then the (thereby invalidated) Executed Activity instance is deleted. (2)<br>If, in turn, the involved Managed Artefact (e.g. SW-module) changes, a new instance of the ManagedArtefact for the new version is created, and the history of the ExecutedActivities remains. |
| responsible | ExecutedProcess Metamodel | Participant | Link to CACM roles not yet provided with the current WEFACT version, currently only individual persons. |

(1) ExecutedProcessModel . ExecutedActivity . startTime/endTime represent the duration of the process execution already performed, not a deadline for a planned future activity.
(2) Executed activities

**WorkflowTool**

ait.ac.at.rcp.wefact.model.types

- subWorkflowToolList: List<WorkflowTool>
- workflowlevel: int
- toolPath. String

A Workflow tool (e.g. a test tool) in WEFACT can be started automatically from the command line with the string given in toolPath.

The **WorkflowTool** mapping to the CACM is depicted in the following table:

**Table 12.** WorkflowTool mapping

| WEFACT | CACM | | comment |
|---|---|---|---|
| element | model | element | |
| subWorkflowToolList | | | Represents the different versions of workflow tools that may be requirement for certain processes. |
| workflowlevel | n/a | | Internal, only usage in WEFACT. |
| toolPath | Managed Artefact | Resource.location, Resource.format | In order for WEFACT to find and execute a tool, the path to the tool must be specified. |

**ProcessObject**

ait.ac.at.rcp.wefactmodel.types

- linkedRequirementObjectlist: List<RequirementObject>
- subProcessObjectList: List<ProcessObject>
- workflowlevel: int
- input: String
- output. String
- workflowTool: VVorkflowTool
- deadline: Date
- workflowStatus: WorkflowStatus
- responsible. String

A ProcessObject can have sub-ProcessObjects. The top-level ProcessObject (without children) corresponds to the WEFACT V-Plan.

The **ProcessObject** mapping to the CACM is depicted in the following table:

**Table 13.** ProcessObject mapping

| WEFACT | CACM | | comment |
|---|---|---|---|
| element | model | element | |
| linkedRequirementObjectlist | SACM Artefact Metamodel | Activity, Artefact | The requirements proven by this V&V activity / V-Plan. |
| subProcessObjectList | | | The V&V activities contained in this V-Plan. |
| workflowlevel | | | Internal only usage in WEFACT. |
| input | Managed Artefact | Resource.location | The SVN sub-directory with the input files. |
| output | Managed Artefact | Resource.location | The SVN sub-directory with the output files. |
| workflowTool | ExecutedProcessModel<br><br>Managed Artefact | UsedTechnique<br><br>Resource.location, Resource.format | Reference to the WEFACT WorkflowTool object. |
| deadline | n/a | n/a | WEFACT specific, no representation in CACM. |
| workflowStatus | n/a | n/a | e.g. ready, success, failed; no representation in CACM. |
| responsible | ExecutedProcess Metamodel | Participant | This references a person (role model not yet implemented in WEFACT, link to CACM roles not yet provided). |

**WEFACT Links**

Links are used to represent traceability between artefacts in WEFACT like RequirementObjects, ProcessObjects and WorkflowTools. They are not 1:1 mapped between the CACM instances and WEFACT but the Open Source interface module controls the establishment of the respective corresponding traceability between the objects in the CACM and those in WEFACT.

**Restrictions**

In the following, the restrictions applying to the WEFACT edition in the 2nd iteration of the AMASS platform are stated:

- The data flow is yet limited to requirements import (no re-export of modified requirements to the UMA process model).
- Consistency of WEFACT activities results with evidence model instances has to be input manually. Or the requirements in WEFACT have to be restricted to a specific structure:
    - Evidences are in a 1:1 mapping with WEFACT top level requirements.
    - Nested (sub-)requirements can be used to subdivide the assurance steps into those activities, which, after successful execution, eventually yield the evidence.
- What is needed for the latter solution is a relation between
    - the activity in the project specific instance of the assurance process model created in EPF-C, and
    - the evidence metamodel instance associated to the solution in the argument metamodel instance.
- Requirements of type contracts/claims as defined in the contract view of the Component Metamodel are not natively supported in the current WEFACT edition.
- The Component Metamodel is currently not supporting sub-requirements; therefore, WEFACT can only map the lowest level of WEFACT requirements in this sub-model. This is, however, basically sufficient as the real assurance steps happen on this lowest level, and higher levels of (compound) requirements are not directly subject to assurance steps.

**WEFACT-CACM/ARTA Workflow**

Here, a short description of the WEFACT workflow is given

- Inputs:
    - Process model, tailored to the project from EPF-C (in UMA notation)
    - Requirements - read from
        - DOORS, or
        - ReqIF, or
        - from CACM, or
        - created in WEFACT
    - Assurance Objects
- Created on the WEFACT user interface:
    - potentially create requirements,
    - potentially alter the project-specific process model,
    - Create and run Assurance Activities with tools assigned in WEFACT
- Outputs:
    - Assurance output files (e.g. test result lists, FMEA sheet, …) (stored in SVN).
    - A statement “PASS” or “FAIL” (within WEFACT, propagated to the requirement).

**Remark:**

The interface module for the functionality for process-based argument generation will be contained in D6.6 [50].

## 3.5 FMVEA Tool (**)

### 3.5.1 Description of Features Implemented in P2 (**)

The following Figure 18 shows the structure of the FMVEA tool including the interfacing with them AMASS platform.

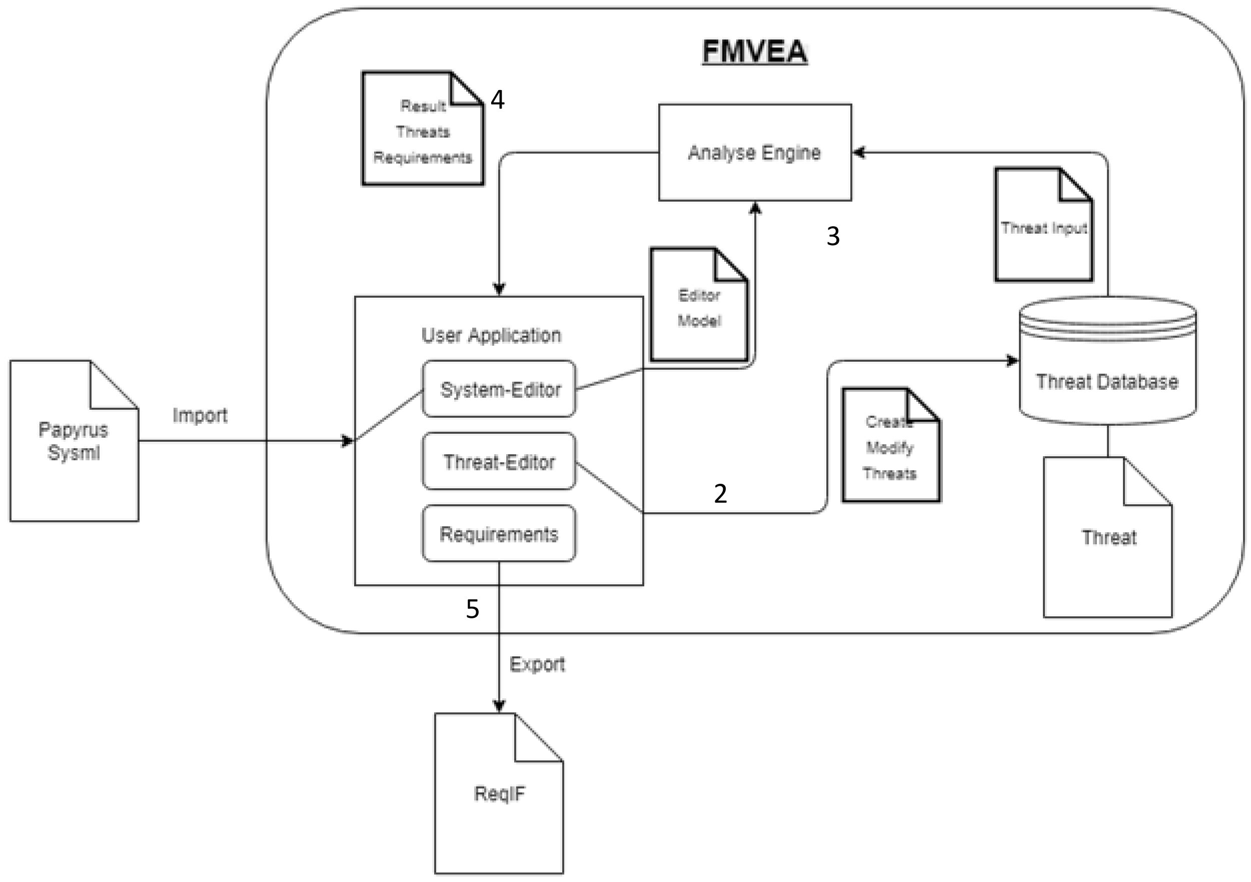


**Figure 18**. Structure of the FMVEA tool including the interfacing with them AMASS platform.

1. Import SysML model or construct it in the FMVEA model editor
2. Definition of threats and failure modes
3. Analyse the current model with respect to threats and failure modes
4. Detected threats and failure modes result in new requirements regarding the analysed system
5. Export the defined requirements into the AMASS platform

### 3.5.2 Interface Module Description (**)

The FMVEA tool provides two interfaces to the AMASS platform:

1. Import of system/component models in CHESS-compatible SYSML format
2. Export of the resulting safety and security requirements regarding the analyzed model.

The following Figure 19 shows the class diagram of the threat class:

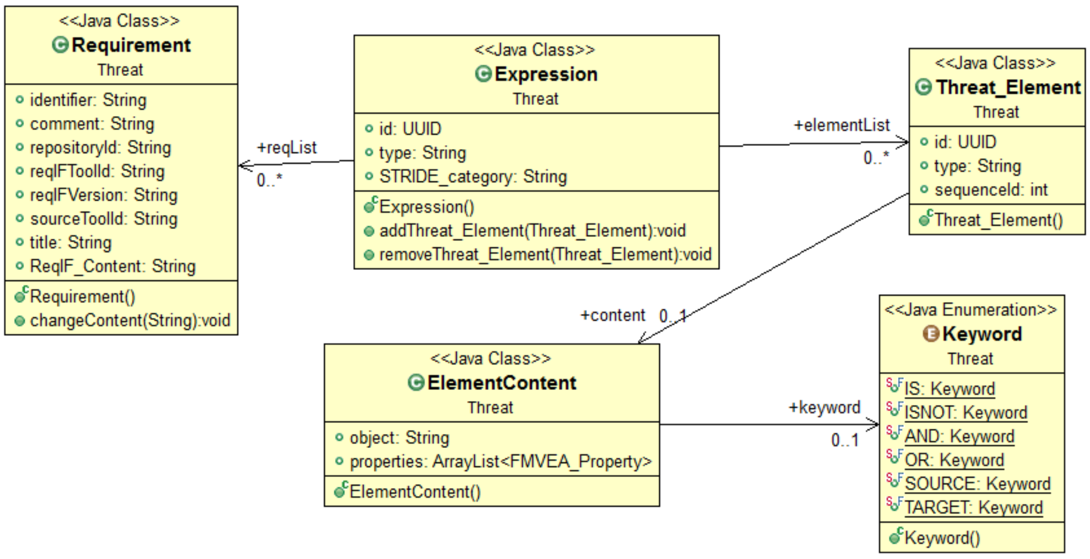


**Figure 19.** Class diagram of the Expression class.

FMVEA provides an Interface in the form of an input field where a user can define new rules. A rule describes if a "Requirement" applies to the analysed system. The defined Requirements apply to the ReqIF standard so they can be imported in AMASS. These rules follow a specific grammar defined with "ANTLR" [53]. Before the rules are added to the database they are checked via NLP (natural language processing).

*Example:*
IF (Environment [type=physical, safe=false] ISPARENTOF Node[type=physical, criticalData=true]) THEN (Requirement1, Requirement2)

The following Figure 20 shows the class diagram of the FMVEA model editor.

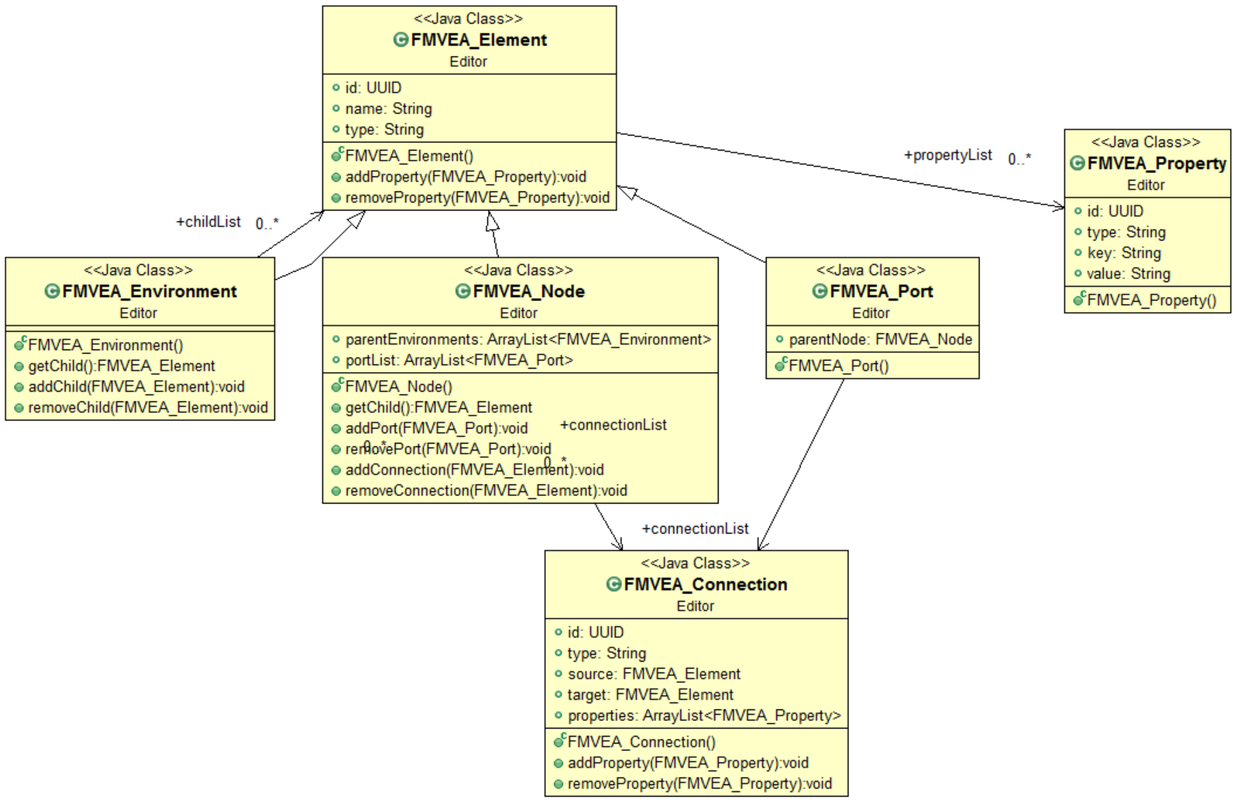
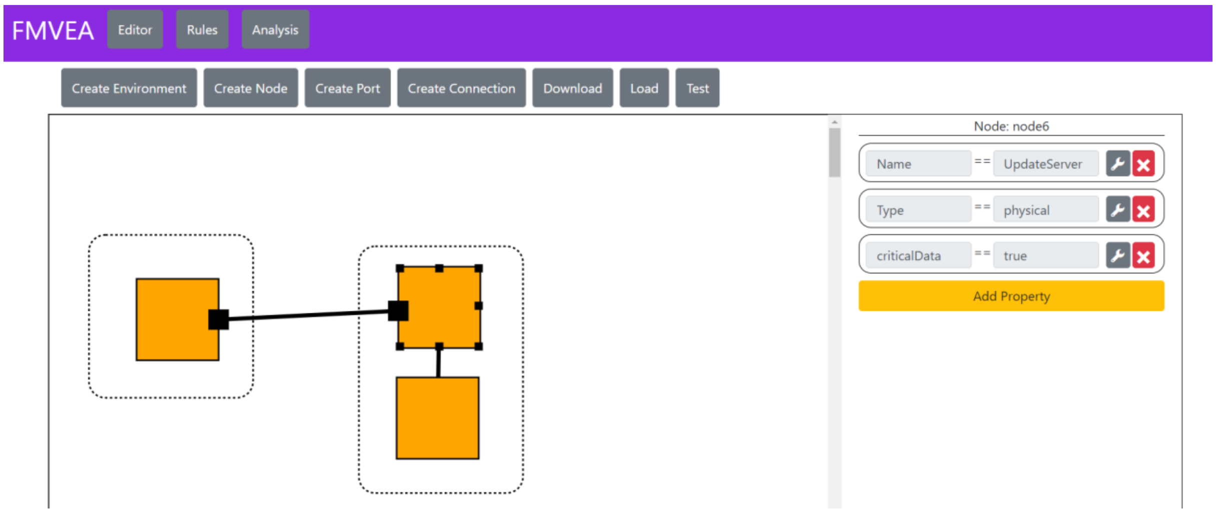


**Figure 20.** Class diagram of the FMVEA model editor



| | | |
|---|---|---|
| Name | == | UpdateServer |
| Type | == | physical |
| criticalData | == | true |

**Figure 21.** User Interface of the FMVEA model editor.

Figure 21 shows the user interface for the model editor. Figure 20 shows the corresponding class diagram for the editor. The modelled instances of the system are analysed and saved in this scheme.

## 3.6 Analytical Network Process (ANP) Tool (**)

### 3.6.1 Description of Features Implemented in P2 (**)

The ANP uses Time Net 4.4 (Stochastic Coloured Petri Nets) tool for system analysis from Technical University of Ilmenau.

Excel VBA will use results from Time NET to automate the ANP matrix generation and analysis for trade-off.

### 3.6.2 Interface Module Description (**)

This tool for now is an external tool. The graphical modelling of the system in Time NET is done manually. Modelling is also possible with an input file in xml format. Therefore, an interface can be developed with CHESS by transforming the CHESS dependability profile into Time NET petri net xml format, in future.

ANP (Analytical Network Process) method will use results of multi-concern co-analysis to perform a metric-based trade-off analysis. ANP will use two existing tools. First tool is Time NET 4.4, which is latest version of tool released in August 2017. Time NET is a software tool for the modelling and analysis of stochastic petri nets with exponentially or non-exponentially distributed times. We will use this tool for manual graphical modelling and analysis of system. Tool is free for research use. Tool and user manual can be downloaded at the site below.

https://timenet.tu-ilmenau.de/template/download
http://www2.tu-ilmenau.de/sse_file/timenet/ManualHTML4/UserManual.html

The results obtained by simulation of the system model in the above tool are written in a file in xml format. Excel VBA (Visual Basic for Application) will be used as a second tool, the results will be imported in excel and with programming in VBA, automatic generation of ANP matrix and analysis of ANP matrix will be achieved. This part is under development for now (6 August 2018) and is expected to be established soon, early August.

Time NET, Coloured Stochastic Petri Nets provides a way for system analysis via simulation. Safety and security Co-analysis in Time NET by combining failure and attack petri nets can be achieved by quantification of attacks. An approach for discrete quantification of attack is proposed in the workshop paper (SAFECOMP 2018) ‘A Quantitative Approach for the Likelihood of Exploits of System Vulnerabilities’. This proposed approach will be worked on for realization after excel VBA for ANP will be realized as mentioned in last paragraph.

The availability, reliability and performance results of system from model simulation in Time NET are imported in excel and with VBA programming, the ANP matrix is automatically generated. The structure of ANP matrix corresponds to hierarchal structure of the system, how component failures and attack failures propagate in the system. And the entries of the ANP matrix signifies the relative impact of these failures on the immediate next hierarchal level. The details of ANP approach is mentioned in detail in D4.3, section 2.1.2.

Final ANP matrix will signify the impact of these failure causes and exploits which are on bottom of hierarchical level to the top goal (safety, security, performance), so we know which failures critical and what design options are available to achieve the required dependability.

One way of Trade-off analysis is that all system architecture design options are evaluated, and best design option can be selected among these, based on evaluation result. This can be achieved by modifying system models in Time NET corresponding to each design option and simulated for results and based on results the desired architecture is selected. This approach is appropriate for very limited design options.

In other case, where we don't have very limited design options, it may not be possible or require too much effort to evaluate all possible design option. In such cases, especially for complex interconnected cyber-physical systems (i.e. for inter-dependability among concerns) ANP is useful because we know with ANP analysis which failure causes or vulnerabilities are critical, and we only need to choose design options corresponding to those failure causes or vulnerabilities which can be verified by rerunning the simulation with the modified design option.

The petri net tool is verified by modelling case study in paper "Safety and availability of railway operation based on state of signalling system. For ANP demonstration this simple case study will be extended to a fictious case to add some complexity.

The model allows us to evaluate the availability and reliability of each component at each hierarchal level, therefore we can easily check if the corresponding requirement is satisfied or not and with ANP what design options can possibly achieve these requirements. Thus, this also enables contract-based analysis.

## 3.7 Concerto FLA extension (**)

ConcertoFLA [31] allows users (system architects and dependability engineers) to decorate component-based architectural models (specified using CHESSML) with dependability-related information, execute Failure Logic Analysis (FLA) techniques, and get the results back-propagated onto the original model. In ConcertoFLA the support for dependability related information is enabled via SafeConcert [29], which is a conceptual metamodel and subset of CHESSML for modelling dependability related information.

Different FLA techniques are available in the literature [37], and can be used at the early stages of the design phase to achieve a robust architecture with respect to linear relationships. ConcertoFLA builds on top of Failure Propagation Transformation Calculus (FPTC) [32]. Similar to FPTC, ConcertoFLA is a compositional technique to qualitatively assess the dependability of component-based systems. ConcertoFLA partially combines and automatizes traditional safety analysis techniques (i.e., FMEA and FTA). ConcertoFLA allows users to calculate the failure behaviour of a component based system at system-level, based on the specification of the failure behaviour of the individual composing components. During the analysis, ConcertoFLA calculates the failure propagation paths and produces their representation according to the specifications of FlaMM meta model (see [33] for FlaMM structure and corresponding XML Schema). In ConcertoFLA terms, a component can act in four different possible ways (1) source of the failure thus generating a failure due to internal fault, (2) sink of the failure thus avoiding the propagation of the external fault (failure in input) through fault tolerance, (3) propagator of the failure, and (4) transformer of the failure into a different type. ConcertoFLA rules are logical expressions, which specify the component's behaviour by describing the input/output relationship.

An initial exploration of the exploitation of ConcertoFLA for enabling safety and security analysis was conducted and documented in D4.7 [27], as methodological guidelines in order to support this exploitation. In D4.3 [25], a more in-depth exploitation is described on conceptual and design level. Moreover, within AMASS, an initial exploration for the exploitation of the failure propagation paths for the generation of Fault Tree (FT) was conducted. The work targeted Use Case 11 as a running example and was accepted for publication at ICRE-2018 [34]. Based on these, following features are implemented in P2 to enable the ConcertoFLA for conducting safety and security analysis and automatically generating FT addressing multi-concern faults/failures. A paper related to the FT generation was recently submitted to ICSRS-2018. Another paper related to co-analysis is expected to be submitted to ICRE-2018. For this reason, not all details are documented.

### 3.7.1 Description of Features Implemented in P2 (**)

Figure 22 shows the overview of the approach for co-analysis via ConcertoFLA – three highlighted steps to perform the co-analysis are enabled by the implementation of the following three features respectively.

- Extension of SafeConcert (dependability profile for CHESSML) for specializing the failure behaviour for security concern.
- Extension of ConcertoFLA to support co-analysis.
- Implementation of a plugin to automatically generating multi-concern Fault Tree (FT).

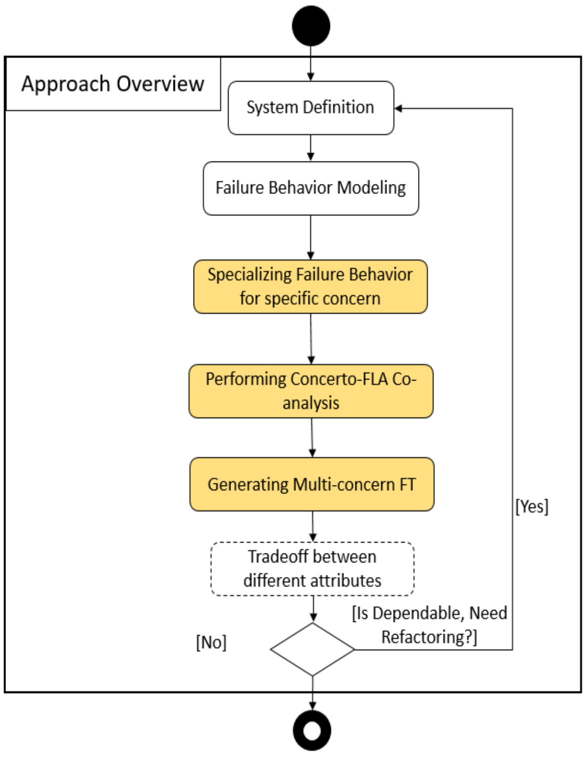

Approach Overview
System Definition
Failure Behavior Modeling
Specializing Failure Behavior for specific concern
Performing Concerto-FLA Co-analysis
Generating Multi-concern FT
[Yes]
Tradeoff between different attributes
[Is Dependable, Need Refactoring?]
[No]

**Figure 22.** Approach overview of co-analysis via ConcertoFLA

Figure 23 shows the meta model extension of SafeConcert, to incorporate the constructs related to security (illustrated using yellow colour). *Threat* is an event or situation that can potentially breach the security and cause harm to an asset. *Attack* rises due to the associated threat and cause the actual breach, if able to exploit a *Vulnerability*. The vulnerability refers to an internal weakness of a system. Threat to security can have three different kinds referring to the breach of security properties. These properties are Confidentiality, Integrity and Availability (CIA). In Figure 23 the *ThreatType* enumeration lists the kind of threats referring to the violation of CIA which are unauthorized access, modification and denial of service respectively. Attack has a relation to the threat, which refers to the security breach it causes, and can have different kinds listed in the *AttackType* enumeration. Vulnerabilities are exploited by the attacks to cause the above-mentioned security breaches – vulnerability can be of different kind and is listed in *VulnerabilityType* enumeration. Different databases and catalogues both domain independent [41] [42] [43] and domain specific [44] are produced, by industrial experts, system engineers and security analysts, who have knowledge of specific domain and systems, for classifying common respective attacks and vulnerabilities in a result of consensus. The enumerations in Figure 23 list attacks to communication satellites missions adopted from CCSDS 350.1-G-2 [44] (attacks are referred to as threat in [44]) and corresponding vulnerabilities which could be exploited by these attacks. These enumerations could be customized for different domains and abstractions referring to different classifications of attacks and vulnerabilities.

In the dependability profile of CHESSML, the above mentioned three security constructs are implemented as a specialization of classical dependability threats (i.e., Faults, Error, and Failure). In CHESSML these classical threats to dependability are modeled as *Transitions* of UML state machine diagram. The attack is a specialized *InternalPropagation* referring to the erroneous transition due to an external fault. The vulnerability is a specialized *InternalFault* referring to the erroneous state transition due to an internal fault. A failure occuring due to these erroneous transitions, has a failure mode and in turn enable the associated threat on the output port of the system. Dependability profile of CHESSML provides the classical abstract failure modes i.e., early, late, valueSubtle, valueCoarse, Commission and Omission. The failure modes and the related security threats enabled by them are as following:

- Commission -> Unauthorized access of a service.
- valueCoarse -> Unauhtorized modification of a service.
- Omission -> Unauthorized denial of service.

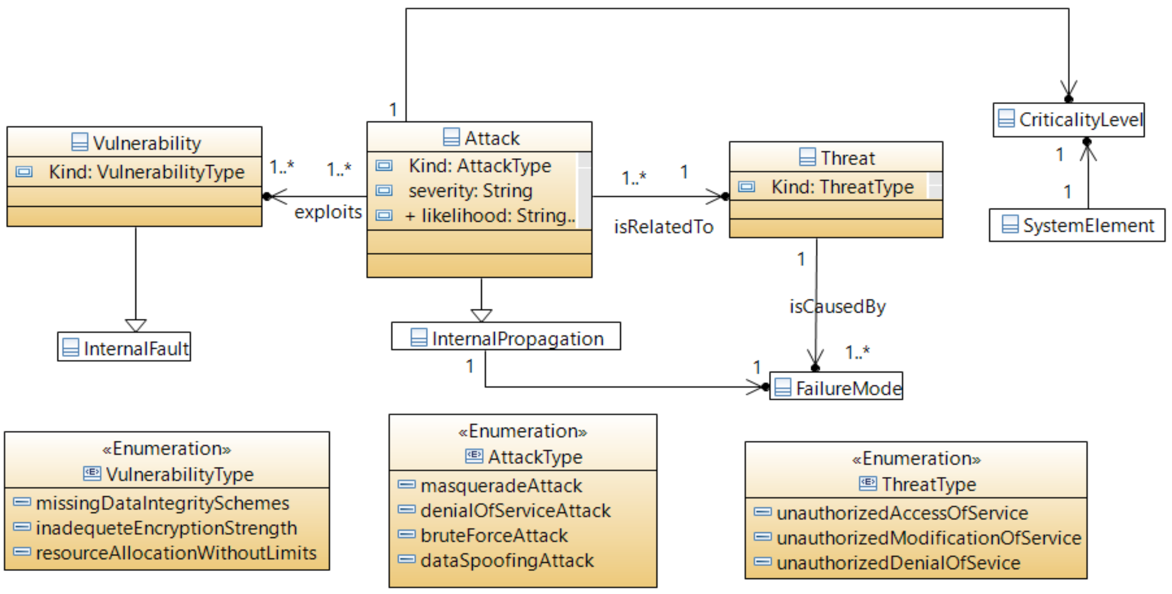


**Figure 23.** Security meta model and its relation to dependability profile of CHESSML

The second feature, which refers to the extension of ConcertoFLA, enables the support for co-analysis. The extension generates failure propagation paths – failures are enriched with the information of security related erroneous transitions (attack and vulnerability) leading to a particular failure along with the information of the specific security breach.

The third feature, which refers to the implementation of multi-concern FT, takes these propagation paths of enriched failures as an input and generate fault trees for each system level failure for each port of the system. Moreover, the plugin enables the support of visualizing and editing these FTs.

### 3.7.2 Source Code Description (**)

The extension of dependability profile of CHESSML (i.e., SafeConcert) discussed in previous section has been implemented in Eclipse using the Papyrus feature for modelling the UML Profiles. Eclipse Modelling Framework (EMF) allows to automatically generate the Java code for such implemented profiles. This profile and the generated code is implemented as an Eclipse plugin for providing the support of modelling the implemented features using CHESS editors.

ConcertoFLA meta-model [31], which has been implemented as an Ecore model using EMF, is extended via Ecore editor to enable the support of specialization of failures. Using EMF the Java code is (re)generated for ConcertoFLA to allow the usage of extension while performing the co-analysis. Moreover, a model to model transformation (CHESSML to FLAMM) is implemented using QVT-operational[5] model transformation language to transform the concern specific specialization of failure behaviour specified via extended dependability profile to the enrichment of failures in the failure

[5] https://projects.eclipse.org/projects/modeling.mmt.qvt-oml

propagation path represented using extended ConcertoFLA meta model. Figure 24 shows ConcertoFLA plugins, the two highlighted plugins refer to the extension of ConcertoFLA meta model and the implemented model to model transformation.

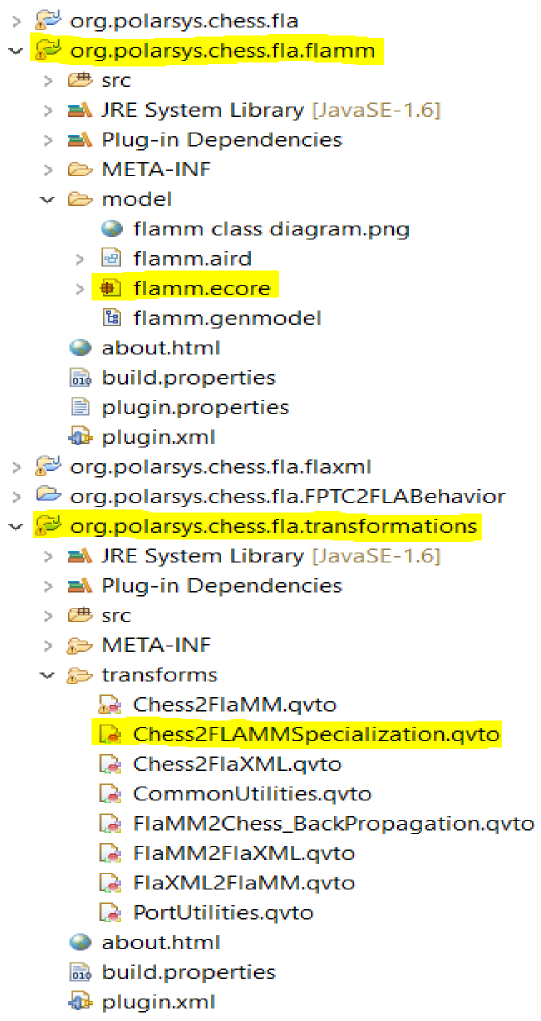
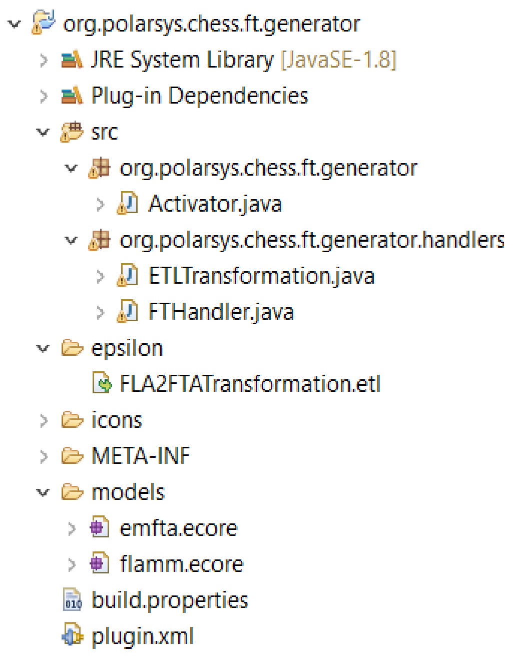

```
> org.polarsys.chess.fla
v org.polarsys.chess.fla.flamm
    > src
    > JRE System Library [JavaSE-1.6]
    > Plug-in Dependencies
    > META-INF
    v model
          flamm class diagram.png
        > flamm.aird
        > flamm.ecore
          flamm.genmodel
      about.html
      build.properties
      plugin.properties
      plugin.xml
> org.polarsys.chess.fla.flaxml
> org.polarsys.chess.fla.FPTC2FLABehavior
v org.polarsys.chess.fla.transformations
    > JRE System Library [JavaSE-1.6]
    > Plug-in Dependencies
    > src
    > META-INF
    v transforms
          Chess2FlaMM.qvto
          Chess2FLAMMSpecialization.qvto
          Chess2FlaXML.qvto
          CommonUtilities.qvto
          FlaMM2Chess_BackPropagation.qvto
          FlaMM2FlaXML.qvto
          FlaXML2FlaMM.qvto
          PortUtilities.qvto
      about.html
      build.properties
      plugin.xml
```

**Figure 24.** ConcertoFLA plugins along with highlights referring to the extended plugins and source

Multi-concern FT generator has been implemented as an Eclipse plugin to allow the generation, visualization and editing of a fault tree from the ConcertoFLA co-analysis results. The plugin implements a model to model transformation (FLAMM to EMFTA[6]) using Epsilon Transformation Language[7] (ETL). Figure 25 shows the source of multi-concern FT generator plugin.

6 https://gitlab.fbk.eu/CPS_Design/EST/

7 https://www.eclipse.org/epsilon/



```
org.polarsys.chess.ft.generator
  JRE System Library [JavaSE-1.8]
  Plug-in Dependencies
  src
    org.polarsys.chess.ft.generator
      Activator.java
    org.polarsys.chess.ft.generator.handlers
      ETLTransformation.java
      FTHandler.java
  epsilon
    FLA2FTATransformation.etl
  icons
  META-INF
  models
    emfta.ecore
    flamm.ecore
  build.properties
  plugin.xml
```

**Figure 25.** Fault tree generator plugin and source

## 3.8 MORETO tool (**)

### 3.8.1 Description of Features Implemented in P2 (**)

The external tool MORETO (Model-based Security Requirements Management Tool) is an Enterprise Architect (EA) plugin and as such compatible with the EA system/component model format. MORETO supports modelling security requirements applying to nodes in a network.

MORETO is an AMASS-external tool for security requirements analysis, allocation, and management using SysML/UML models. It is applicable to any system that can be modelled in SysML/UML (e.g. cyber-physical production systems CPPS). The tool is implemented in Enterprise Architect MDG technology and supports manual as well automatic security requirements generation and allocation.

MORETO provides a model editor but can as well import (EA conformant) SysML models. The models, which capture system facts in layers of abstraction, are exploited for security analysis and documentation. The model editor allows seamlessly navigating through layers with different level of detail. MORETO supports security analysis on high level as well as on technical level, and requirements can be generated manually or automatically. **Figure 26** shows the options for requirements generation in MORETO.

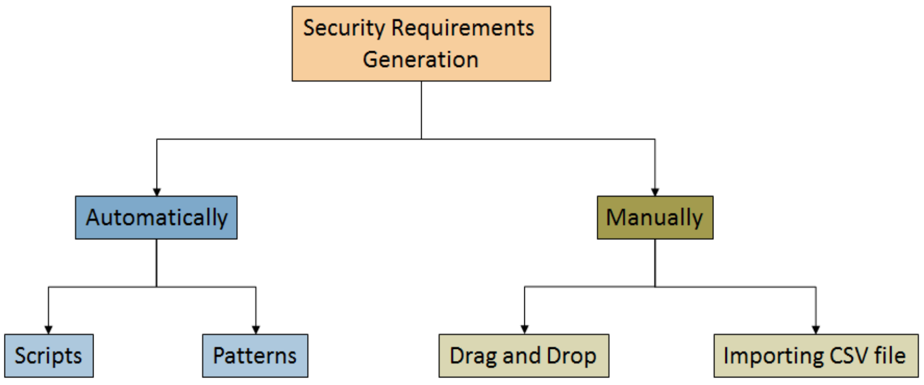


**Figure 26.** Options for requirements generation in MORETO

The security requirements are an integral part of MORETO in the form of the requirements diagram. Moreover, the tool offers the possibility to import additional requirements and functions for exporting them to different formats. For creating security requirements manually, MORETO provides the security requirement diagram, where the user can specify a particular security requirement, and a drag and drop mechanism to assign it to the respective element in the model. As an alternative MORETO allows the import of csv files.

For automatic security requirements generation, MORETO offers two different ways:

- **Patterns** are a feature provided by Enterprise Architect to generate a set of components which are integrated to solve an abstract problem. It is the task of the pattern user to modify the pattern elements to meet the specific demands.
- **Scripts** executed in Enterprise Architect have access to the currently open model and are a powerful tool for querying and updating the model in situations that would otherwise require time consuming and repetitive GUI tasks.

The automatic security requirements generation is based on expert knowledge encoded in MORETO usualy based the prescriptions of standards. For realizing the standards, patterns were used with MORETO; currently implementations for IEC 62443 [48] and IEEE 1686 [49] are available from AMASS Case Study 1 [46].

MORETO supports the system modelling process in four different diagrams:

- the Block Definition Diagram (BDD) for network elements,
- the Internal Block Diagram (IBD) for detailed modelling,
- the Dataflow Diagram (DFD) for Threat Modelling, and, finally,
- the Requirement diagram for security requirements.

The modelling process in MORETO can be done on three different layers, the External Layer, the Intermediate Layer, and the Internal Layer.

The **External Layer** defines a general system schematic or network architecture and allows a user to drag and drop common network components into the workspace in order to generate the complete network topology. Figure 27 gives an example for the External Layer.

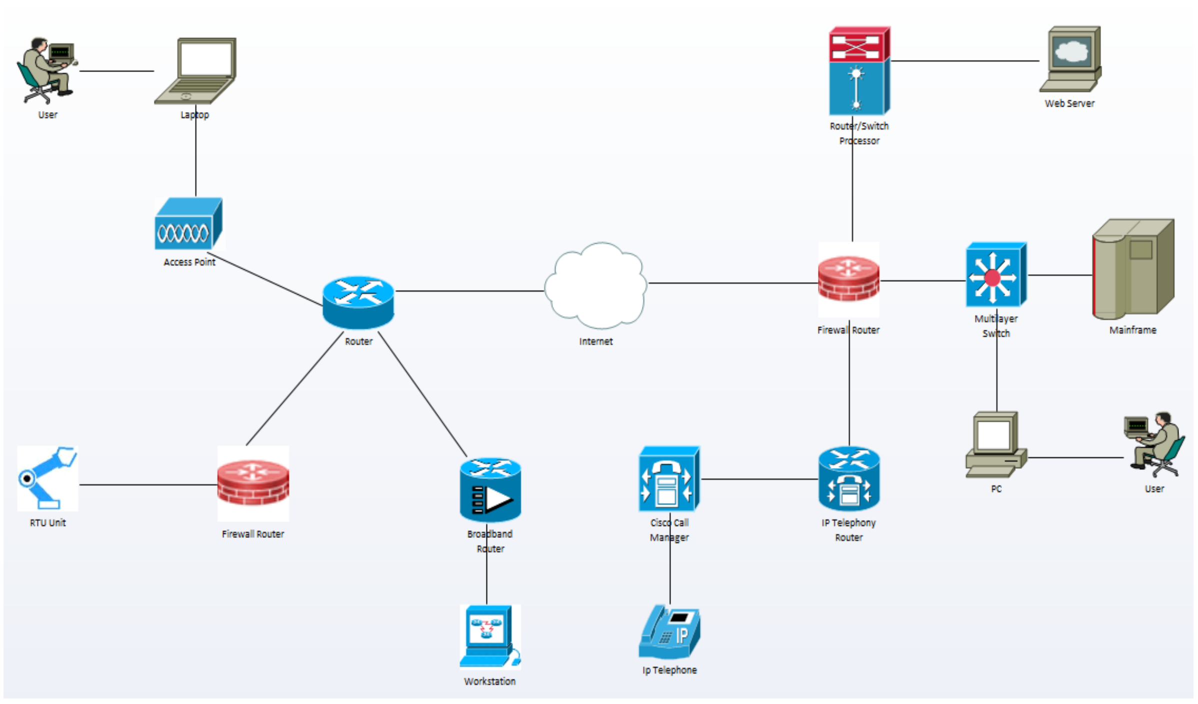


**Figure 27.** Example for the MORETO External Layer

The **Intermediate Layer** defines the internal structure of each component inserted in the external layer and allows a user to describe the interactions between different components of the network topology using the Internal Block Diagram. Figure 28 presents an exemplary Intermediate Layer.

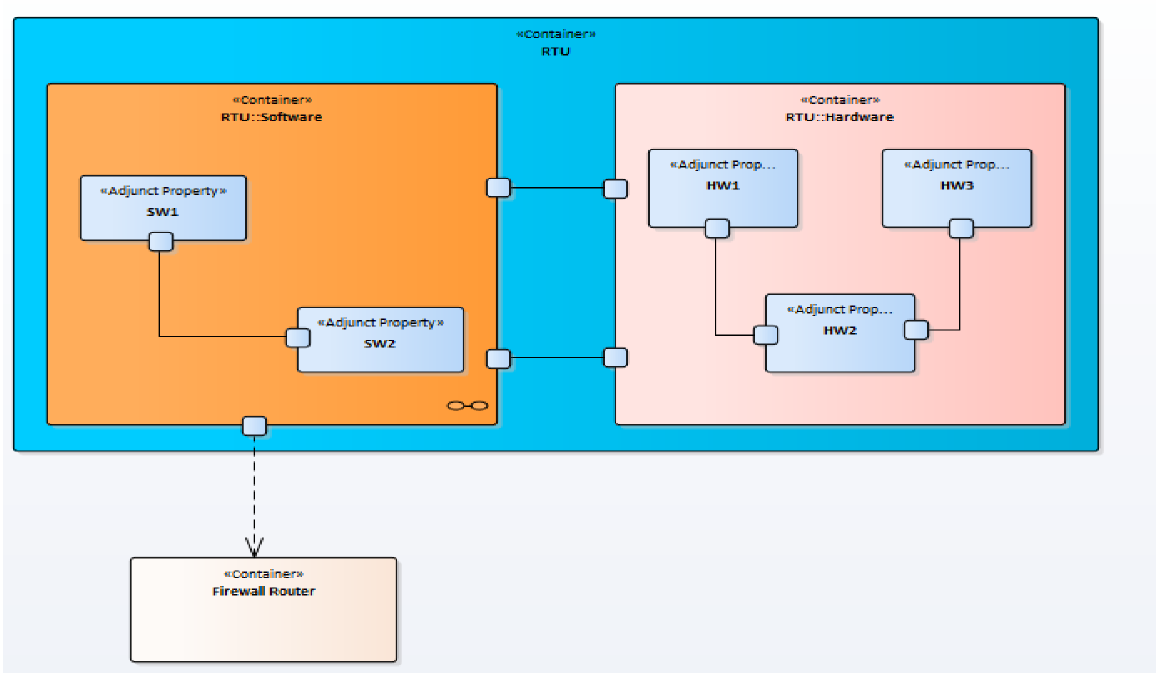


**Figure 28**. Example for the MORETO Intermediate Layer

The **Internal Layer** provides more information about the internal details of the intermediate level components and allows a user to define the internal structure in arbitrary details, or to decompose a block into parts or subsystems. This layer uses the internal block diagram. An example for the Internal Layer can be seen in Figure 29.

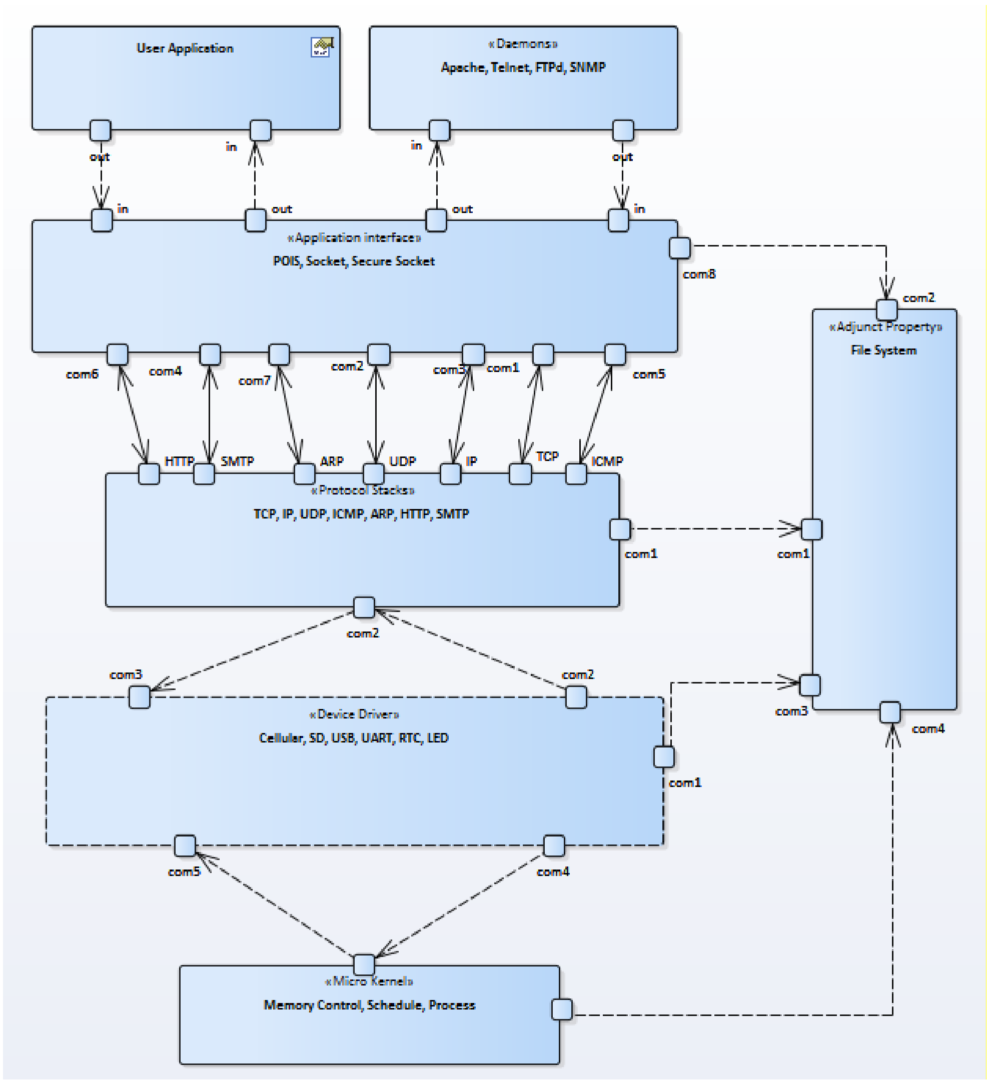


**Figure 29.** Example for the MORETO Internal Layer.

More details about the concrete implementation for the application in CS1 can be found in D1.5 [46].

The current version is designed for one iteration of defining security requirements for a system. If the system model is changed outside MORETO, information about the modifications is needed when re-importing the model from CHESS in order to restrict the next iteration of analyses to the modified parts. This extension is a future aspect, whose implementation can be done in a use case in the final months of the AMASS project or in future projects beyond AMASS.

### 3.8.2 Interface Module Description (**)

MORETO is an Enterprise Architect (EA) Plugin. As such, it can provide its output (requirements and system/component model) in the EA-specific dialect of SysML. CHESS, as the AMASS-internal modelling language and environment, uses a different SysML dialect as model interchange format. The required model transformation at the interface between MORETO and CHESS/Papyrus is accomplished with the tool ModelBus.

ModelBus [40] was developed by the System Quality Center at Fraunhofer FOKUS Berlin. It provides a framework for interoperability and allows the MORETO model export to CHESS/Papyrus and vice versa by supporting Client and Server versions for model transformations, see Figure 30.

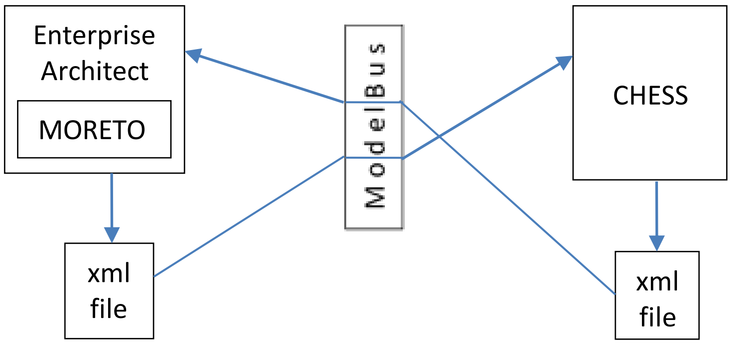


**Figure 30.** Model transformation at the interface between MORETO and the AMASS platform.

## 3.9 Medini Analyzer (**)

Medini analyze is an external tool to perform safety and security analyses - either combined or individually - based on system models. These system models can be modelled within the tool using SysML or imported from various sources (other SysML tools like Rational Rhapsody, or Enterprise Architect but also BOM lists and IP design files). The software is implemented based on the Eclipse Platform as a Rich Client. All analyses can be performed on different abstraction levels, from high level system or function models down to hardware and even ip design (chip design) models.

The tool offers various well-known security analysis methods, for example TARA, Attack Trees and Security FMEA. All methods are flexible with respect to used risk graphs (e.g. EVITA, CVSS/CRSS) or processes (e.g. JASO TP-15002) but also to specific thread identification and risk assessment methods (e.g. STRIDE based).

### 3.9.1 Description of Features Implemented in P2 (**)

Medini is treated as an external tool to the AMASS platform, thus is not a part of the platform but can integrate with it. Papyrus SysML models - which might have been created with AMASS tools - can be imported (and analysed) using the ANSYS SCADE Architect as a bridge tool. SCADE Architect is a separate tool which is based on Papyrus itself (while medini is using its own SysML metamodel and dialect to cope with the fact that safety engineers are typically no system engineers and interested in a certain simplified view on the system). Medini offers an import from SACDE Architect so via that migration path Papyrus SysML models – and thus AMSSS models – can be utilized in security analyses, as depicted by Figure 31.

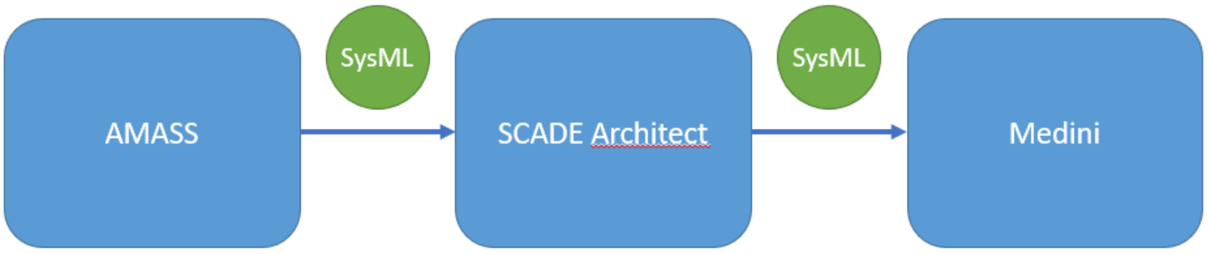


**Figure 31.** Import path between AMASS and medini.

### 3.9.2 Interface Module Description (**)

The above-mentioned path from AMASS system models to models that can be analysed in medini is simple and straight. Any Papyrus model that has been created using AMASS Papyrus based tools can be loaded into the SCADE Architect (SCA) standalone tool. SCA is a rich client itself and available as a separate tool within the SCADE/medini tool landscape. One imported, the model can be either pushed from SCA to medini or "pulled" from medini (imported into a medini own representation). This import can be repeated multiple times to update the system model in medini while keeping any analysis results.

# 4. Conclusion (*)

The **first** (**Core**) prototype, described in D4.4 [26], contained functions for Dependability Assurance Modelling, namely the Assurance Case Editor supporting "Assurance case specification", and the AMASS Core edition of CHESS supporting contract modelling with OCRA.

This previous deliverable D4.5 [51] described the WP4 related part of the **second** AMASS prototype iteration (**P1**) and contained implementations or tools covering already all functional blocks related to WP4. Not all tools were yet integrated with the AMASS platform and the remaining ones needed still manual integration. For the letter, integration was planned for the third iteration P2.

For **Dependability Assurance Modelling**, enhanced functions were added in P1 in the (at that time already integrated) Assurance Case Editor and by integrating also EPF (Eclipse Process Framework) in the AMASS platform. With this complete set of functionalities, no further extensions were necessary in the third iteration (P2).

**Contract-based Multi-Concern Assurance** was also supported in P1 by integrated tools: Additional respective features of the Assurance Case Editor, and new multi-concern contract features in the CHESS integrated framework provided these functions.

Enhanced functions for contract-based Multi-Concern Assurance were provided in P2, namely two different approaches for Contract-based Trade-off Analysis implemented in tools: a CHESS tool extension using parameterized architectures for trade-off analysis, and a prototypic implementation of an Analytical Network Process (ANP) using the tool TimeNET4.4 of University Illmenau/Germany.

In iteration P1, the third functional block of AMASS WP4, **System Dependability Co-analysis/Assessment**, was mostly supported by external, not yet integrated tools: Medini Analyzer for safety analysis and risk assessment and with a yet prototypic cybersecurity analysis part, Safety Architect, and the tool AMT2.0 (Analogue Monitoring Tool). The detailed description of AMT2.0, however, is contained in the WP3 documents.

In P2, additional tools and tool extensions for System Dependability Co-analysis/Assessment were introduced: A mature commercial version of Medini Analyzer, the safety and cybersecurity co-analysis tool FMVEA (Failure Modes, Vulnerabilities and Effects Analysis), the cybersecurity analysis and requirements allocation tool MORETO (Model-based Security Requirements Management Tool), and Concerto-FLA (an Extension of the Concerto tool allowing Failure Logic Analysis).

WEFACT has been integrated in AMASS in iteration P1 and controls the execution of assurance process steps. For WP4 the main feature is to enable combined safety and security analyses by coupling separate analysis tools for both quality attributes. However, WEFACT is capable of controlling assurance processes of any kind and their dependencies.

In the third iteration P2, several innovations regarding multiconcern assurance have been achieved. This comprises the automatic, standards-based generation of security requirements from an architectural model like with MORETO, and the development of two methods for trade-off analysis between different quality attributes: A contract-based approach with CHESS, and another one relying on an Analytical Network Process. Another innovation is the introduction of the partly automated safety-security-co-analsysis tools FMVEA and Medini Analyzer, and the functionality for automatically generating assurance case arguments from the modelled assurance processes.

The multi-concern approach developed in WP4 addresses in particular the project objective O2, for instance by providing multi-concern contracts or multi-concern trade-off analyses. Moreover, by providing interfaces to the AMASS platform for external multi-concern assurance tools, the work described in this deliverable is also in line with the project objective of seamless interoperability as stated in O4.

All these advances contribute significantly to the AMASS project goals; in particular the increase in automation addresses G1 by improved design efficiency for CPS products, and the standards-based generation of requirements contributes to G3 by reducing the assurance and certification/qualification risks of new CPS products.

The next steps are now the validation of the tools in the case studies and the evaluation of the resulting use case-specific multi-concern assurance processes.

# Abbreviations and Definitions

For the convenience of the reader, the following table also contains definitions common to the whole AMASS project which are contained in the AMASS glossary (deliverable D2.2 [17]).

| *Abbreviation* | *Explanation* |
|---|---|
| AMT | Analog Monitoring Tool (a property-based monitoring tool for analog systems) |
| API | Application Programming Interface |
| ARTA | AMASS Reference Tool Architecture |
| BVR | Base Variability Resolution (Language for managing variability) |
| CACM | Common Assurance and Certification Metamodel |
| CCL | Common Certification Language |
| CDO | Connected Data Object |
| CHESSML | CHESS Modelling Language |
| CPS | Cyber-Physical Systems |
| DOORS | Dynamic Object-Oriented Requirements System |
| ECSEL | Electronic Components and Systems for European Leadership |
| EMF | Eclipse Modelling Framework |
| EPF-C | Eclipse Process Framework - Composer |
| ETL | Epsilon Transformation Language |
| FLA | Failure Logic Analysis |
| FMEA | Failure Modes and Effects Analysis |
| FMVEA | Failure Modes, Vulnerabilities and Effects Analysis |
| FPTC | Failure Propagation Transformation Calculus |
| FT | Fault Tree |
| FTA | Fault Tree Analysis |
| GMF | Graphical Modeling Framework |
| GSN | Goal Structuring Notation |
| HARA | Hazard Analysis and Risk Assessment |
| OCRA | Othello Contracts Refinement Analysis (a tool for checking refinement of contracts specified in a linear-time temporal logic) |
| OMG | Object Management Group |
| OSLC | Open Services for Lifecycle Collaboration |
| RCP | Rich Client Platform - an Eclipse add-on framework allowing the development of Eclipse applications |
| ReqIF | Requirements Interchange Format (XML based standard of OMG) |
| SACM | Structured Assurance Case Metamodel |
| SBVR | Semantics of Business Vocabulary and Rules |
| SiSoPL | Security-informed Safety-oriented Process Line |
| SPL | Software Product Line |
| SVN | Subversion |
| SySML | Systems Modelling Language |
| TARA | Threat Analysis and Risk Assessment |
| TRL | Technology Readiness Level |
| UMA | Unified Method Architecture |

| | |
|---|---|
| UML | Unified Modelling Language |
| URL | Uniform Resource Locator |
| V&V | Verification and Validation |
| WEFACT | Workflow Engine for Analysis, Certification and Test |
| WP | Work Package |
| XML | eXtensible Markup Language |
| Xtext | open-source software framework for developing programming languages and DSLs |

# References (*)

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

[2] The SafeCer project https://artemis-ia.eu/project/40-nsafecer.html

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

[4] BVR Base Variability Resolution – implementation of OMG CVL (Common Variability Language) http://www.omgwiki.org/variability/doku.php .

[5] OMG - SACM - Object Management Group version 1.1, 2015 http://www.omg.org/spec/SACM/1.1

[6] Origin Consulting GSN Community Standard Version 1 (2011)

[7] Graphical Modelling Project (GMP) http://www.eclipse.org/modeling/gmp/

[8] Eclipse Modelling Framework (EMF) https://www.eclipse.org/modeling/emf/

[9] Eclipse Process Framework (EPF) http://www.eclipse.org/epf/

[10] Epsilon Transformation Language http://www.eclipse.org/epsilon/doc/etl/

[11] Xtext http://www.eclipse.org/Xtext/

[12] Eugenia http://www.eclipse.org/epsilon/doc/eugenia/

[13] CDO http://www.eclipse.org/cdo/

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

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

[16] AMASS D2.1 Business cases and high-level requirements, 28 February 2017.

[17] AMASS D2.2 AMASS Reference Architecture (a), 30 November 2016.

[18] AMASS D2.3 AMASS reference architecture (c), 30 September 2017.

[19] AMASS source code https://services.medini.eu/svn/AMASS_source/[8]

[20] AMASS Platform – Prototype Core User Manual[9]

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

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

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

[24] AMASS D4.2 Design of the AMASS tools and methods for multi-concern assurance (a), 30 June 2017.

[25] AMASS D4.3 Design of the AMASS tools and methods for multi-concern assurance (b), 31 January 2018

[26] AMASS D4.4 Prototype for multi-concern assurance (a), 31 January, 2017

[27] AMASS D4.7 Methodological guide for multiconcern assurance (a), December 2017

[28] AMASS D6.2 Design of the AMASS tools and methods for intra/cross domain reuse (a), 31 October 2017.

[8] The AMASS SVN code repository is open to AMASS partners with the same credentials as the SVN document repository. In case that people outside the project need access, please contact the AMASS Project Manager (alejandra.ruiz@tecnalia.com)

[9] The current User Manual is a draft document; the final version of the manual will be integrated in D2.5 AMASS User guidance and methodological framework (m31).

[29] L. Montecchi and B. Gallina. SafeConcert: a Metamodel for a Concerted Safety Modeling of Socio-Technical Systems. 5th International Symposium on Model-Based Safety and Assessment (IMBSA), Trento, Italy, September 2017.

[30] SysML v1.4 Specification Release, September 2015. http://www.omgsysml.org/specifications.htm

[31] B. Gallina, E. Sefer and A. Refsdal, "Towards Safety Risk Assessment of Socio-Technical Systems via Failure Logic Analysis," *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, Naples, 2014, pp. 287-292.

[32] M. Wallace. Modular architectural representation and analysis of fault propagation and transformation, vol. 141, no. 3, pp. 53–71, 2005.

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

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

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

[36] http://www.omg.org/spec/MARTE

[37] L. Grunske, J. Han, "A Comparative Study into Architecture-Based Safety Evaluation Methodologies using AADL's Error Annex and Failure Propagation Models", 11th IEEE High Assurance Systems Engineering Symposium, pp. 283–292, Nanjing, China, 3-5 Dec., 2008.

[38] https://services.medini.eu/svn/AMASS_collab/WP4/D4.5_in_progress/WP4-Requirements_Iteration2

[39] AMASS D3.6 Prototype for Architecture-Driven Assurance, 31 August 2018

[40] ModelBus https://www.modelbus.org (access July 2018)

[41] Common Attack Pattern Enumeration and Classification, https://capec.mitre.org/

[42] Common Vulnerabilities and Exposures, https://cve.mitre.org/

[43] Common Weakness Enumeration, http://cwe.mitre.org/index.html

[44] CCSDS 350.1-G-1, Report Concerning Space Data Systems Standards, Security Threats Against Space Missions, December 2015

[45] AMASS D2.5 AMASS user guidance and methodological framework, 31 October 2018

[46] AMASS D1.5 AMASS demonstrators (b), 21 April 2018

[47] AMASS D6.3 Design of the AMASS tools and methods for intra/cross domain reuse (b), 30 July 2018.

[48] IEC 62443 Industrial communication networks – Network and system security, 2009 ff

[49] IEEE 1686-2013 – IEEE Standard for Intelligent Electronic Devices Cyber Security Capabilities, 2013

[50] AMASS D6.6 Prototype for cross/intra-domain reuse (c), October 2018

[51] AMASS D4.5 Prototype for multi-concern assurance (b), October 2017

[52] http://www.eclipse.org/proposals/polarsys.chess/

[53] www.antlr.org

[54] M. A. Javed and B. Gallina. Safety-oriented Process Line Engineering via Seamless Integration between EPF Composer and BVR Tool. In 22nd International Systems and Software Product Line Conference (SPLC), Sept 10-14, Gothenburg, Sweden, in press. ACM Digital Library, 2018.

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

[56] J. P. Castellanos Ardila, B. Gallina and F. Ul Muram. Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models. Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Prague, Czech Republic, August 29-31, 2018.

[57] AMASS D4.8 Methodological guide for multiconcern assurance (b), October 2018