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 (b)
D4.5

Work Package:	WP4 Multi-Concern Assurance
Dissemination level:	PU = Public
Status:	Final
Date:	31 October 2017
Responsible partner:	Thomas Gruber (AIT)
Contact information:	Thomas.gruber@ait.ac.at
Document reference:	AMASS_D4.5_WP4_AIT_V1.0

PROPRIETARY RIGHTS STATEMENT

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

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	AIT Austrian Institute of Technology GmbH
Alejandra Ruiz	Tecnalia Research & Innovation
Barbara Gallina, Irfan Sljivo	Maelardalen Hoegskola (MDH)
Stefano Puri	Intecs (INT)

Reviewers

Names	Organisation
Marc Sango (Peer-reviewer)	All4Tec
Garazi Juez (Peer-reviewer)	Tecnalia Research & Innovation
Cristina Martínez (Quality Manager)	Tecnalia Research & Innovation
Jose Luis de la Vara (TC review)	Universidad Carlos III de Madrid

TABLE OF CONTENTS

Executive Summary **6**

1. Introduction **7**

2. Implemented Functionality **9**

2.1 Scope 9

2.2 Implemented Requirements - Overview 9

2.2.1 Requirements implemented in the Assurance Case Editor in OpenCert 13

2.2.2 Requirements realized in EPF-Composer & BVR Tool 14

2.2.3 Requirements implemented in CHESS 14

2.2.4 Requirements implemented in WEFACT 15

2.3 Installation and User Manuals 17

2.3.1 Internal tools 17

2.3.2 External tools 18

3. Implementation Description **19**

3.1 Assurance Case Editor from OpenCert 19

3.1.1 Description of Implemented Features 19

3.1.2 Source Code Description 20

3.2 EPF-Composer Tool 22

3.2.1 Description of Realized Features 22

3.2.2 Source Code / Interface Description 22

3.3 CHESS Tool 22

3.3.1 Description of Implemented Features 22

3.3.2 Source Code Description 23

3.4 WEFACT Tool 24

3.4.1 Description of Implemented Features 24

3.4.2 Interface Module Description 26

4. Conclusion **33**

5. Outlook and Next Steps **34**

Abbreviations and Definitions **36**

References **38**

Appendix A: Additional CHESS Functionalities **40**

List of Figures

Figure 1. AMASS Building blocks 7
Figure 2. Tool modules for Assurance Case Management Component 19
Figure 3. Tool module from Contract Management Component 20
Figure 4. Assurance Case Specification plugins 22
Figure 5. Contract profile supporting modelling of concerns 23
Figure 6. Argument Generator plugin source 24
Figure 7. WEFACT Activity Diagram 25
Figure 8. Relation between WEFACT low level activity results and the CACM 27
Figure 9. Relation modelled between WEFACT activity results on high level and the CACM evidences 27
Figure 10. The WEFACT Metamodel 28
Figure 11. Initiating the argument-fragment generation (Step 1) 41
Figure 12. Selecting the source analysis context (Step 2) 42
Figure 13. Selecting the destination assurance case folder on the CDO repository (Step 3) 43
Figure 14. Generation successfully completed with argument-fragments for each block 44
Figure 15. An example of the generated argument-fragment 45

List of Tables

Table 1. Functionality to be implemented in iterations 2 & 3 of the AMASS platform according to D4.4 [28] 10
Table 2. Requirements implemented in the second prototype of the AMASS platform (P1) 11
Table 3. Requirements implemented in the Assurance Case Editor 13
Table 4. Requirements [partly] implemented in WEFACT 16
Table 5. Available installation documentation and user manuals for the tools implemented in iteration 2 18
Table 6. WefactProject mapping 29
Table 7. RequirementObject mapping 30
Table 8. WorkflowTool mapping 31
Table 9. ProcessObject mapping 31
Table 10. Functionality to be implemented in the third iteration (P2) of the AMASS platform 35

Executive Summary

This deliverable, D4.5 Prototype for multi-concern assurance (b), is the second output of the task T4.3 *Implementation for Multi-Concern Assurance*. Based on the results from task T2.2 *AMASS Reference Tool Architecture and Integration*, 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 support the requirements for tooling developed in WP4. Task 4.3 is being 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 [43] 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 supported the following two functional areas:

- Argumentation Editor
- Argument Patterns Editor

The release at hand is the second prototype iteration, which extends 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].

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 WEFACT workflow engine as an external tool is integrated via an open source interface module and described in detail in the document at hand. This includes source code references of the open source interface module, information about the technology used and a description of mapping between the WEFACT-internal database and the AMASS CACM.

Other important parts of D4.5 are:

- Executable of the WEFACT tool [32].
- WEFACT user manual and installation instructions [33].
- Source code of the WEFACT-CACM interface module (in [18]).

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 will extend 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 will correspond 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 will be 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.

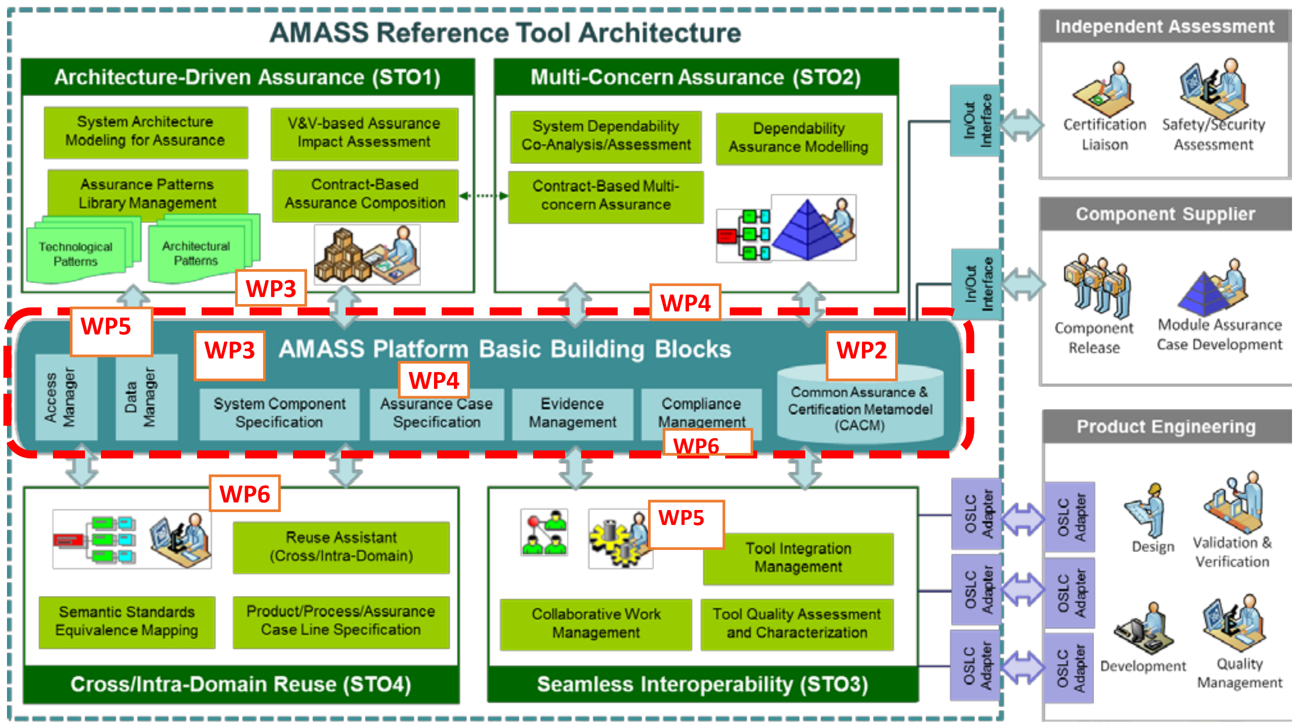

AMASS Reference Tool Architecture
Architecture-Driven Assurance (STO1)
System Architecture Modeling for Assurance
V&V-based Assurance Impact Assessment
Assurance Patterns Library Management
Contract-Based Assurance Composition
Technological Patterns
Architectural Patterns
Multi-Concern Assurance (STO2)
System Dependability Co-Analysis/Assessment
Dependability Assurance Modelling
Contract-Based Multi-concern Assurance
WP3
WP4
WP5
WP3
AMASS Platform Basic Building Blocks
WP2
WP4
Access Manager
Data Manager
System Component Specification
Assurance Case Specification
Evidence Management
Compliance Management
Common Assurance & Certification Metamodel (CACM)
WP6
WP6
Reuse Assistant (Cross/Intra-Domain)
Semantic Standards Equivalence Mapping
Product/Process/Assurance Case Line Specification
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 Building blocks

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 planned 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 have been 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 will be integrated in a comprehensive toolset operating at TRL5. Functionalities specific for multi-concern assurance developed for the second prototype will be 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 case studies for the first prototype was described in D1.1 [22]. For the second prototype, implementations applying WEFACT in CS1 and CS3 are under elaboration, for CS3, moreover, the application of FMVEA in iteration 3 (P2) is planned.

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 current implementation. This refers to the following functionalities:

- Support for contract-based multi-concern assurance 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.

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 functionality and describes the interface and its mapping to the CACM in detail. Furthermore, it references the interface module source code as well as user and installation manuals.

Other important parts of D4.5 deliverable are:

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

[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

2. Implemented Functionality

2.1 Scope

This second prototype of the Multiconcern assurance module has the purpose to complete 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 have been 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), 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 to combine tools for analyses w.r.t. different concerns is in the focus.

The following tools, described in deliverable D4.2 [25], are envisaged to be integrated in the third iteration of the AMASS platform (prototype P2):

- Medini Analyzer - supports the assurance process workflow and allows safety analyses and, in a prototypic version, security analyses.
- Safety Architect – supports "System Dependability Co-analysis/Assessment" for safety and security, in particular a combination of FTA and ATA.
- AMT2.0 - supports "Contract-Based Multi-concern Assurance" by generating monitors for observing properties of nodes a network.

The following tool is currently (Oct. 2017) being re-developed as an Eclipse RCP application and will be integrated with the AMASS platform in the third iteration (prototype P2):

- FMVEA - supports "System Dependability Co-Analysis/Assessment".

The Farkle tool, which verifies learning algorithms based on volume testing, supports product assurance for a very specific case; its use is currently being investigated in a use case (CS3) but, for the time being, not planned to be integrated with the AMASS platform (neither second nor third iteration).

2.2 Implemented Requirements - Overview

From the requirements point of view this phase focuses on a set of AMASS requirements as defined in deliverable D2.1 [16].

According to D4.4 [28], the following functionality shall be implemented in iterations 2 and 3:

Table 1. Functionality to be implemented in iterations 2 & 3 of the AMASS platform according to D4.4 [28]

Function name	Description
Argumentation architecture	The system shall be able to edit an argument architecture associated with a system and/or component.
Semi-automatic generation of product arguments (1)	The system shall reduce efforts of manual creation of product-based assurance case arguments. This could be done by enabling semi-automatic generation of product-based arguments-fragments.
Assurance case status report	The system shall provide the capability for querying the assurance case in order to detect: 1) undeveloped goals, and 2) fallacies.
Assurance case structure navigation	The system shall let the user browse the assurance case structure. Note: in case GSN-like modelling elements are used, this requirement may be translated as follows: The system shall let the user navigate from top-level assurance case overview to the nested assurance case fragments that are encapsulated within modules.
Provide guidelines for argumentation patterns	The system shall be able to provide guidelines to use and instantiate argument patterns (concerning safety and security) presented in the actual assurance case.
Compliance map generation from argument evidences	The system shall be able to detect when a claim about a requirement from a standard (compliance claim) is supported by an evidence and to generate the compliance indicator in a transparent way.
Formal validation of assumptions and context when arguments modules are connected	The system shall be able to indicate the validation of assumptions contained in argument modules every time the modules are connected and/or modified.
Provide quantitative confidence metrics about an assurance case in a report	The system could produce a status report indicating a quantitative confidence metric for the assurance case.
Provide guidelines for argumentation	The system shall be able to provide guidelines about the assurance case edition based on the system/component development phase status.
The AMASS tools must support specification of variability at the argumentation level (2)	The system shall provide the capability for modelling a multi-concern and multi-context assurance case. Note: variability modelling could be 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.
Argumentation import/export	The system shall be able to import/export argumentations to SACM.

(1) An implementation of this function is documented in D6.5 [31]. Here in D4.5, the focus is on multi-concern, i.e. on those cases where contracts regarding safety as well as contracts regarding security are applied.
(2) In addition, this requirement is in conjunction with WP6. Again, here the focus of consideration is on variability w.r.t. different quality attributes in the sense of multi-concern.

Table 2. Requirements implemented in the second prototype of the AMASS platform (P1)

Requirement No	Name	Description	Tool
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
WP4_ACS_007	Argumentation import/export	The system could be able to import/export argumentations to SACM.	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
WP4_ACS_010	Composition of the overall argument	The system should provide the capability of generating a compositional assurance case argument.	OpenCert
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 trade-off process	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 the most optimal ones.	OpenCert
WP4_CMA_001	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.	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.	CHESS
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. (1)	WEFACT, Medini Analyzer, Safety Architect
WP4_SDCA_003	The system shall allow combinations of safety and security analysis	The system shall allow combinations of safety and security analysis. (2)	WEFACT, Medini Analyzer, Safety Architect
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

Requirement No	Name	Description	Tool
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
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
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). 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 and BVR tool

(1) WEFACT allows combining V&V activities (e.g. calls to test tools) in one complex activity.
(2) In this second iteration P1, the combination of safety and security analysis is achieved by combined assurance activities in WEFACT, in which a separate safety and a separate security analysis tool are called in parallel or sequentially (can be configured to the needs). In iteration 3 calls to combined tools are foreseen (e.g. FMVEA).
(3) Partially implemented.

In particular w.r.t. WEFACT, the implementation covers also requirements belonging to work packages other than WP4; that is why the following table of requirements implemented in the second iteration of the AMASS platform contains not only WP4 requirements. The column "Requirement No" refers to the IDs in the deliverable D2.1 [16].

Each tool together with the implementation done so far that implements requirements are 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 3.

Table 3. 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 previously but have been improved in this second iteration. Same with others, now the Assurance Case Editor has some of the functionality covered but it could be modified in the third iteration, after getting feedback from the validation task and from the case studies' implementers.

WP4_ACS_001: Assurance case edition

This requirement was previously covered in Prototype Core.

WP4_ACS_002: Argumentation architecture

This requirement is focused on previously commented functionality, Assurance case structure navigation. 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 previous prototype (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 covered in the previous prototype (Prototype Core). There has 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 this second iteration. 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). Before 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 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 this iteration (Prototype P1). In deliverable D4.2 [26] the "dependency relationship" has been presented. The new implementations have tried to cover this new dependency relationship concept.

2.2.2 Requirements realized in EPF-Composer & BVR Tool

As it was recalled in D4.2 [27], EPF Composer is the tool that implements the EPF (Eclipse Process Framework) 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 [30], EPF-C is currently being 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. 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 [30]. D4.7 [29], instead, will include in-depth guidance on how to benefit from such integration in the context of multi-concern (co-) assessment.

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 [40]) and then analysed (WP4_SDCA_001 requirement). In particular, (a subset of) MARTE [41] 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 Appendix A.

In the context of AMASS, the extension of CHESS to cover the modelling and analysis of security aspects is currently under investigation, in particular by considering what is already available from other modelling tools (e.g. SafetyArchitect provided by All4Tec), trying to understand if specific integration at modelling language and/or tool can be realized. For instance, modelling of security aspects could be provided in CHESS by extending the current CHESS dependability profile (see Appendix A).

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.

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 2. Nevertheless, they were elaborated at least conceptually and documented in D4.2 [26]. Here, a description is given in Appendix A: Additional CHESS Functionalities.

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.

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

Table 4. 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). This visualization could be done via compliance maps (matrix) or via arguments aimed at justifying the satisfaction of the requirements coming from the standards.

1) Partly implemented.
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 a WP3 requirement.

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.

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.

2.3 Installation and User Manuals

2.3.1 Internal tools

The steps necessary to install the second prototype are exhaustively described in the AMASS User Manual for all the AMASS building blocks [20] and will not be repeated here. That document contains all required steps and document references to set up the tools. A pre-packaged distribution is being supplied in the second iteration of the AMASS platform.

In summary, that document is a user manual of the second AMASS tool prototype implementation. The users can find there the installation instructions, the tool environment description, and the functionalities for not just the assurance case specification but also for the other basic building blocks.

2.3.2 External tools

External tools have a stub description in the AMASS User Manual and possibly manuals on the tool provider website. Table 5 depicts an overview on available installation documentation and user manuals for the tools implemented in iteration 2.

Table 5. Available installation documentation and user manuals for the tools implemented in iteration 2

Tool	Available Installation Documentation and User Manual
CHESS	Contained in: https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP1/AMASS_PrototypeP1_UserManual.doc
EPF-Composer	Contained in: https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP1/AMASS_PrototypeP1_UserManual.doc
WEFACT	https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP1/WEFACT_UserManual.docx https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP1/WEFACT_Installation_Guide.docx

CHESS and EPF-Composer are part of the Core platform; their description is therefore contained in the general AMASS user manual. WEFACT is an external tool; it is an Eclipse RCP application that is started outside Eclipse as a separate executable file. Its documentation is available at the tool provider AIT; nevertheless, for ease of use, its documentation including User Manual and Installation Guide is provided to the AMASS project participants in the same directory as the AMASS-internal tools.

3. Implementation Description

3.1 Assurance Case Editor from OpenCert

3.1.1 Description of Implemented Features

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 [26].

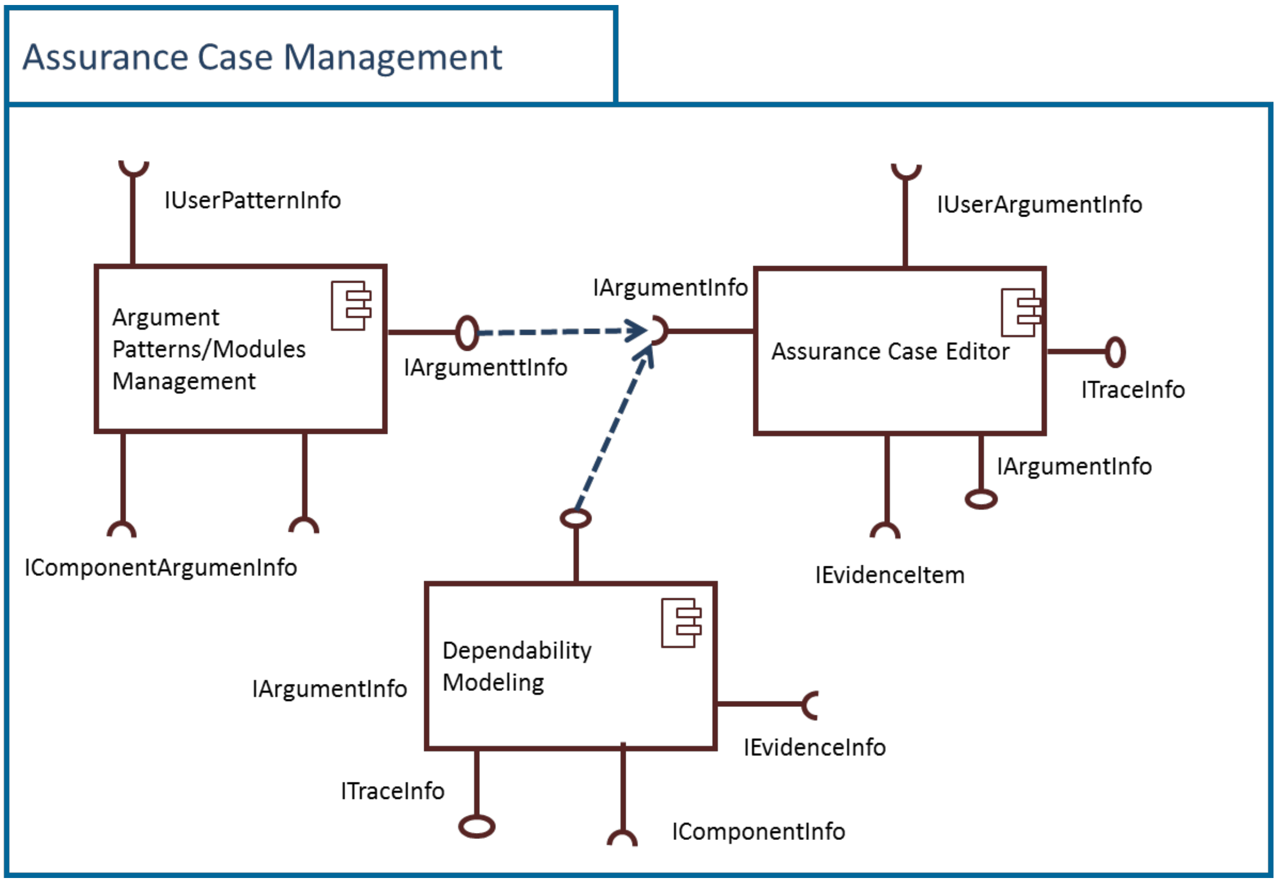


Figure 2. Tool modules for Assurance Case Management Component

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.

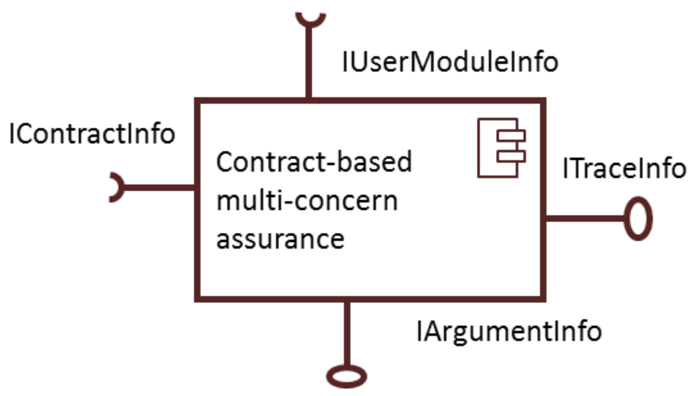


Figure 3. Tool module from Contract Management Component

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 2, 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 Source Code Description

The source code of the first AMASS prototype can be found in the source code SVN repository at https://services.medini.eu/svn/AMASS_source.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.
- 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 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 plugins 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 4 illustrates the list of plugins described above.

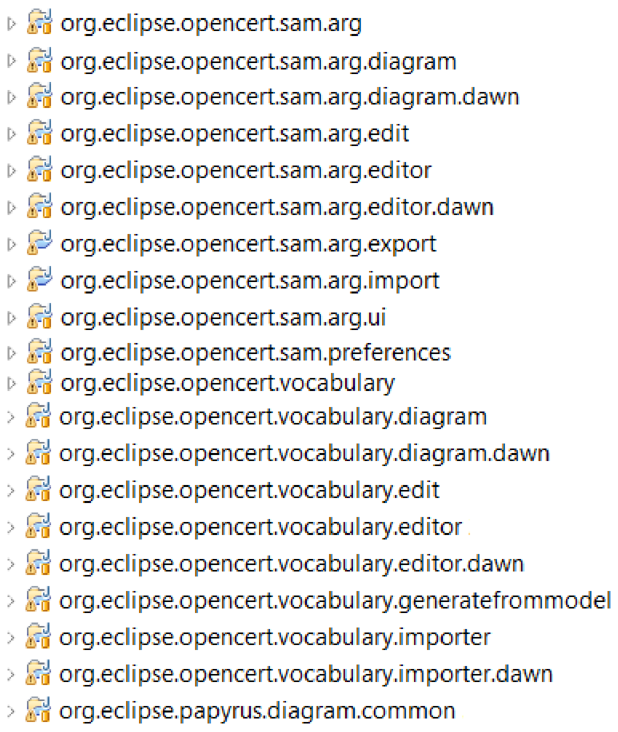

org.eclipse.opencert.sam.arg
org.eclipse.opencert.sam.arg.diagram
org.eclipse.opencert.sam.arg.diagram.dawn
org.eclipse.opencert.sam.arg.edit
org.eclipse.opencert.sam.arg.editor
org.eclipse.opencert.sam.arg.editor.dawn
org.eclipse.opencert.sam.arg.export
org.eclipse.opencert.sam.arg.import
org.eclipse.opencert.sam.arg.ui
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 4. Assurance Case Specification plugins

3.2 EPF-Composer Tool

3.2.1 Description of Realized Features

For AMASS, no specific features were realized in EPF-C, but the original tool was used as it is available on http://www.eclipse.org/epf/composer_architecture/. In addition, detailed documentation can be found on this web site. In AMASS WP4, EPF-C is used for creating and tailoring the project-specific assurance workflow model starting from a standard-specific model.

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 but the software has not been developed within AMASS.

3.3 CHESS Tool

3.3.1 Description of Implemented Features

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 5), 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).

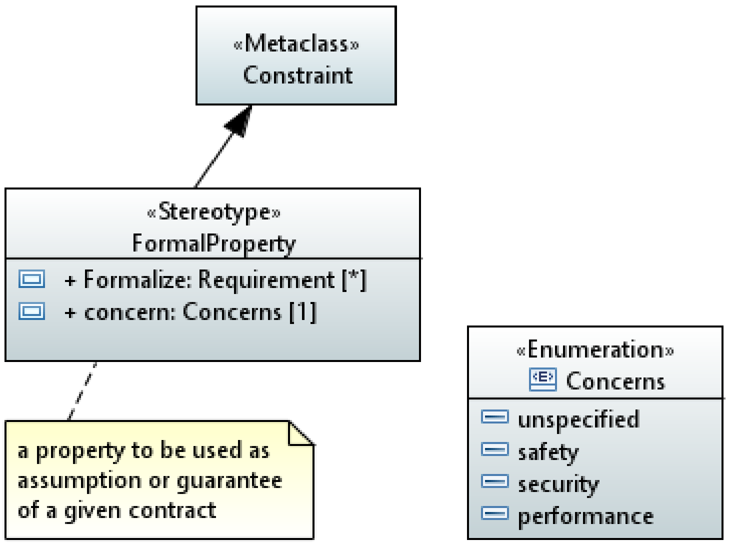


Figure 5. Contract profile supporting modelling of concerns

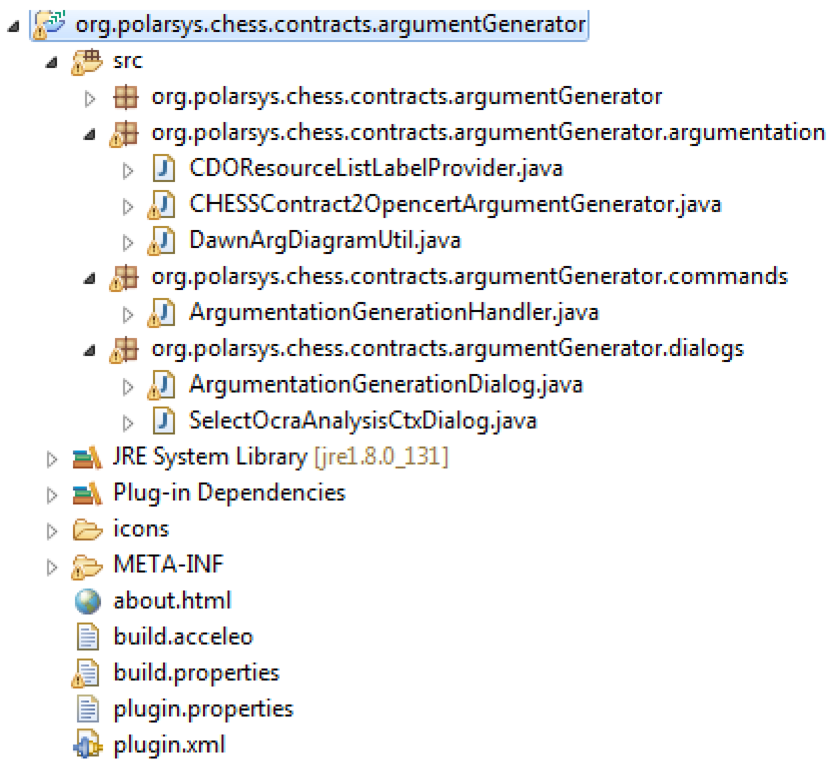
CHESS functionality has been extended with an Argument Generator plugin that utilises the assurance and concern specific information attached to the contracts and facilitates generation of argumentation fragments for each component in the system model. As the different contracts and related assurance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

3.3.2 **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[2] 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 6. 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.

[2] 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 6. Argument Generator plugin source

3.4 WEFACT Tool

3.4.1 Description of Implemented Features

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 7 shows the WEFACT Activity Diagram.

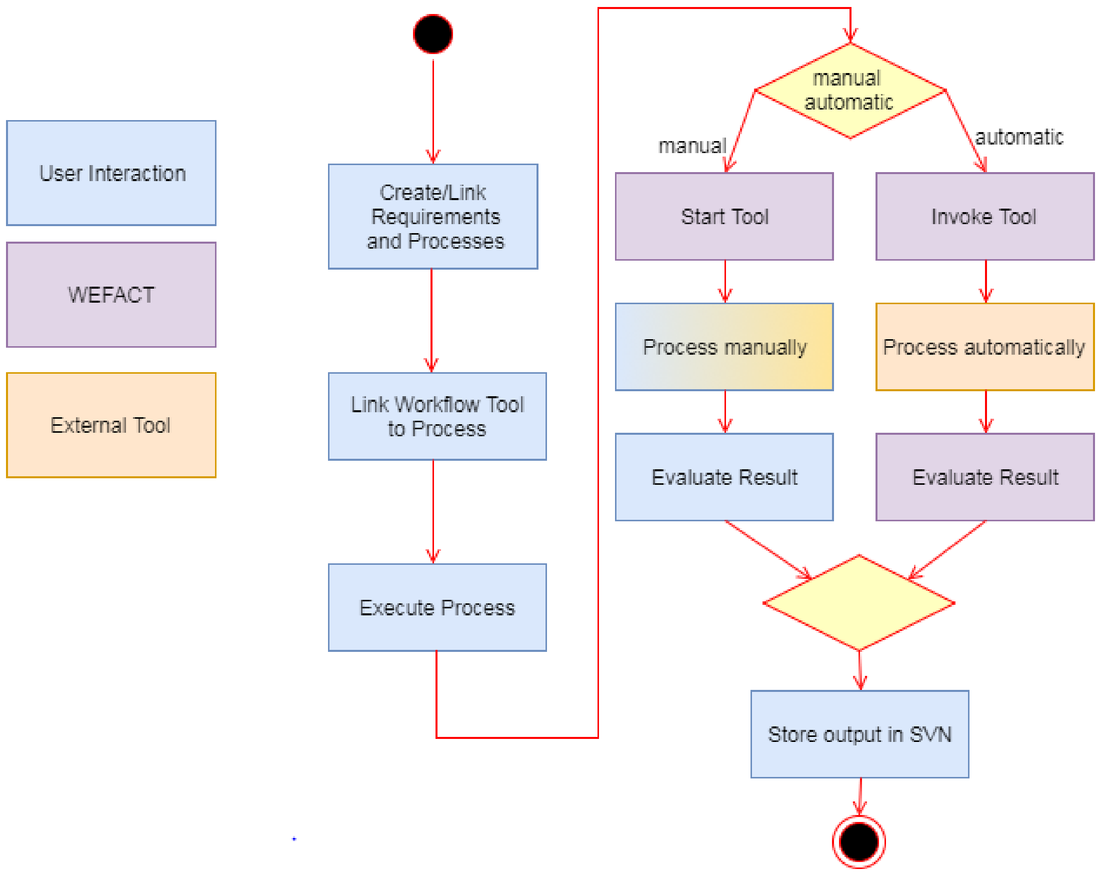


Figure 7. WEFACT Activity Diagram

3.4.1.3 Centralized SVN Storage

The biggest advantage of storing artifacts 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 artifacts that allows all partners to work collectively on a common solution rather than on independent ones.

3.4.1.4 EPF 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 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 8 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.

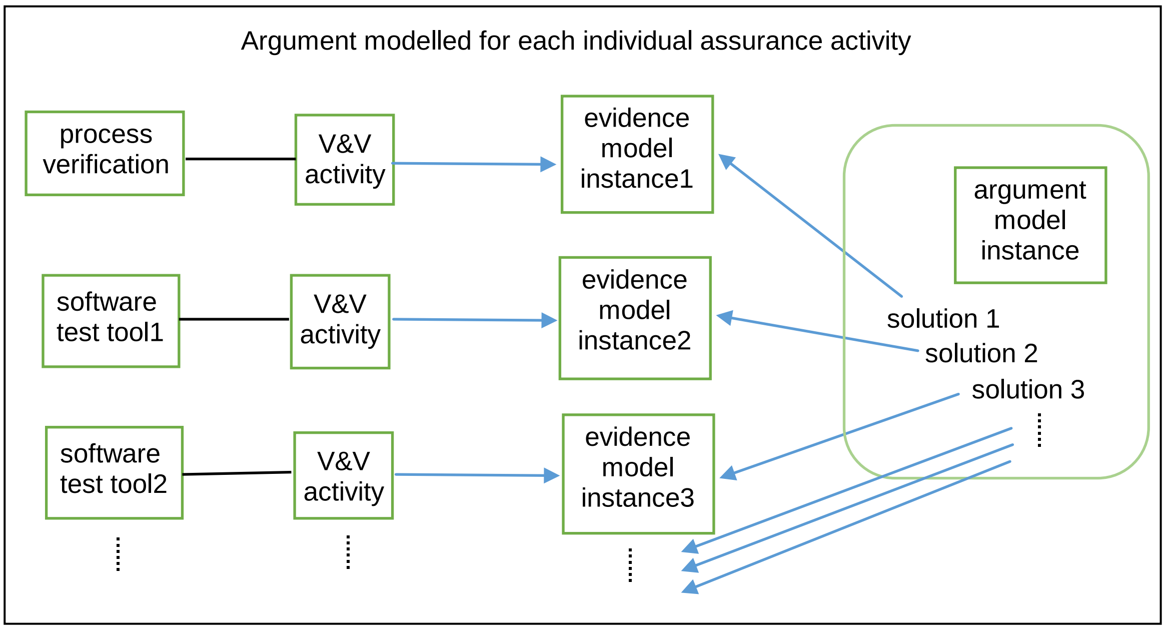


Figure 8. Relation between WEFACT low level activity results and the CACM

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 9 shows graphically an example for the respective relation.

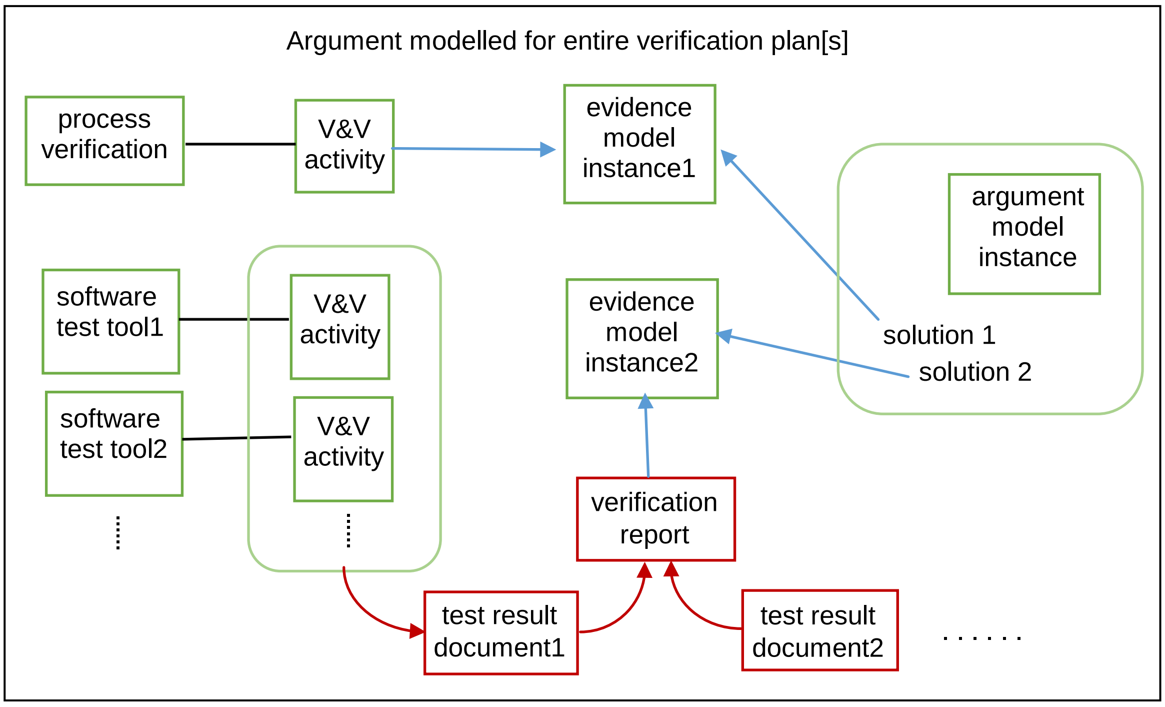


Figure 9. Relation modelled between WEFACT activity results on high level and the CACM evidences

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

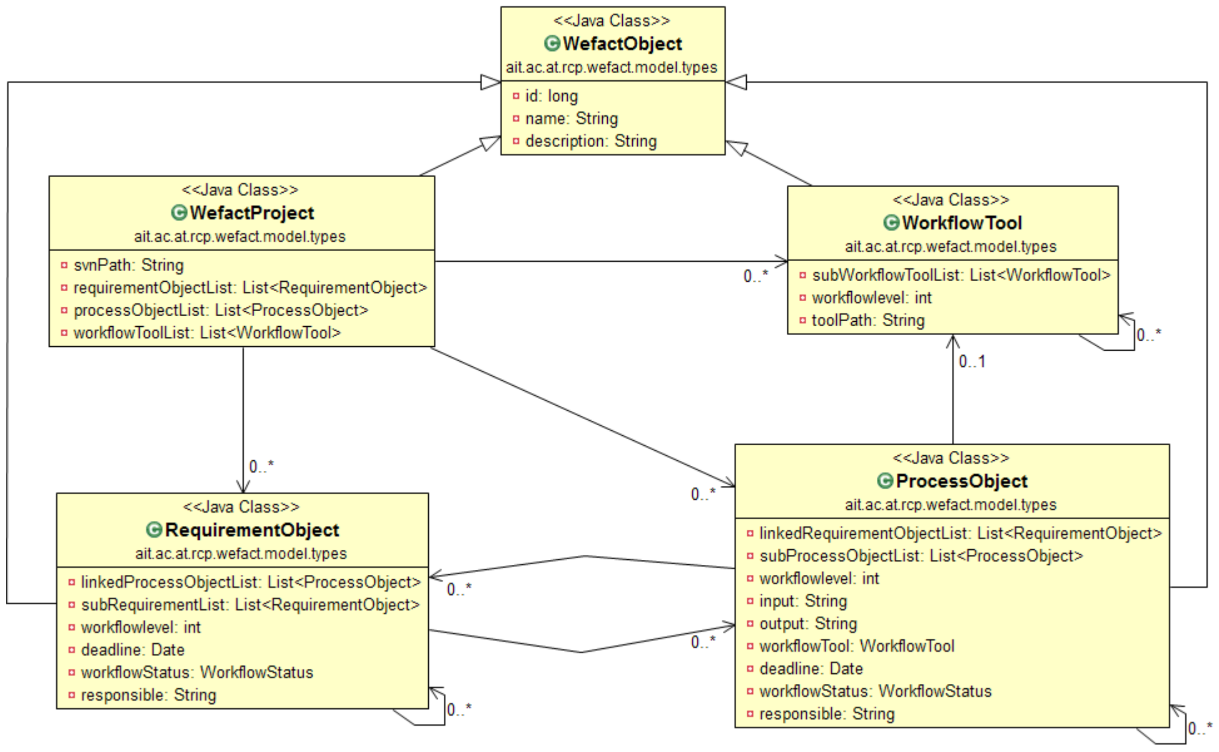

<<Java Class>>
WefactObject
ait.ac.at.rcp.wefact.model.types
id: long
name: String
description: String
<<Java Class>>
WefactProject
ait.ac.at.rcp.wefact.model.types
svnPath: String
requirementObjectList: List<RequirementObject>
processObjectList: List<ProcessObject>
workflowToolList: List<WorkflowTool>
<<Java Class>>
WorkflowTool
ait.ac.at.rcp.wefact.model.types
subWorkflowToolList: List<WorkflowTool>
workflowlevel: int
toolPath: String
<<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..1

Figure 10. 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 6. 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 b) basic product req.: Component MM	task, work product Requirement	The requirements associated with the assurance project. The list contains toplevel as well as sub requirements, process and product requirements. 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 7. 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. 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) 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 8. 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 9. 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 Managed Artefact	UsedTechnique 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).

4. Conclusion

The **first** (**Core**) prototype, described in D4.4 [28], 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 deliverable described the WP4 related part of the **second** AMASS prototype iteration (**P1**) and contains implementations or tools, respectively, covering all functional blocks related to WP4. Not all tools are yet integrated with the AMASS platform and the remaining ones need, for the time being, manual integration in P1. Nevertheless, the full range of functional blocks has been implemented, and for the remaining ones, integration is planned for the third prototype P2.

For Dependability Assurance Modelling, enhanced functions are now available in the (integrated) Assurance Case Editor and by integrating EPF in the AMASS platform.

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

In iteration P1, the third functional block of AMASS WP4, System Dependability Co-analysis/Assessment, is mostly supported by external, not yet integrated tools: Medini Analyzer, Safety Architect, and – belonging equally to WP3 and WP4 - the tool AMT2.0 (Analogue Monitoring Tool).

WEFACT takes a specific positionThis tool is 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.

One of the proposed tools, FMVEA, is currently (Oct. 2017) being re-developed as an Eclipse RCP application. It will be available in the third iteration P2 and then integrated with the AMASS platform, too.

5. Outlook and Next Steps

For the third iteration of the AMASS platform an extension of the prototype for multi-concern assurance with workflow functions is planned in collaboration with WP6 by enhancing the tool WEFACT [21] with capabilities supporting process based argument generation.

Another multi-concern assurance tool is planned to be integrated in the third prototype iteration (P2), the Eclipse RCP application FMVEA (Failure Modes, Vulnerabilities and Effects Analysis), which is currently being developed based on a prior incomplete prototypic implementation. FMVEA allows combined safety and security analysis based on empiric data.

A third external tool is expected to be integrated in the third iteration, namely Medini Analyzer, which in a prototypic version will support combined safety and security analysis including the generation of combined Fault and Attack trees.

According to the requirements defined in D2.1 [16], the functionality in Table 10 shall be implemented in the third iteration (P2).

Table 10. Functionality to be implemented in the third iteration (P2) of the AMASS platform

Function name	Description	Requirement
Argumentation architecture	The system shall be able to edit an argument architecture associated with a system and/or component.	WP4_ACS_002
Semi-automatic generation of product arguments	The system shall reduce efforts of manual creation of product-based assurance case arguments. This could be done by enabling semi-automatic generation of product-based arguments-fragments.	WP6_PPA_002
Assurance case status report	The system shall provide the capability for querying the assurance case in order to detect: 1) undeveloped goals, and 2) fallacies.	WP4_ACS_011
Assurance case structure navigation	The system shall let the user browse the assurance case structure. Note: in case GSN-like modelling elements are used, this requirement may be translated as follows: The system shall let the user navigate from top-level assurance case overview to the nested assurance case fragments that are encapsulated within modules.	High-level requirement 4.7
Provide guidelines for argumentation patterns	The system shall be able to provide guidelines to use and instantiate argument patterns (concerning safety and security) presented in the actual assurance case.	WP4_ACS_004
Compliance map generation from argument evidences	The system shall be able to detect when a claim about a requirement from a standard (compliance claim) is supported by an evidence and generate the compliance indicator in a transparent way.	WP6_CM_010
Formal validation of assumptions and context when arguments modules are connected	The system shall be able to indicate the validation of assumptions contained in argument modules every time the modules are connected and/or modified.	WP4_ACS_012
Provide quantitative confidence metrics about an assurance case in a report	The system could produce a status report indicating a quantitative confidence metric for the assurance case.	WP4_ACS_013
Provide guidelines for argumentation	The system shall be able to provide guidelines about the assurance case edition based on the system/component development phase status.	WP4_ACS_006
The AMASS tools must support specification of variability at the argumentation level	The system shall provide the capability for modelling a multi-concern and muti-context assurance case. Note: variability modelling could be 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.	WP4_CMA_001
Argumentation import/export	The system shall be able to import/export argumentations to SACM.	WP4_ACS_007

The WP6 related requirements are to be implemented in collaboration with WP6.

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)
ANP	Analytical Network Process (Approach for multi-concern trade-off analysis based on Markov models)
API	Application Programming Interface
ARTA	AMASS Reference Tool Architecture
ATA	Attack Tree Analysis
BVR	Base Variability Resolution (Language for building SPL)
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
DSL	Domain-Specific Language
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 Transform Logic
FTA	Fault Tree Analysis
GMF	Graphical Modeling Framework
GSN	Goal Structured Notation
HARA	Hazard Analysis and Risk Assessment
ISO	International Organization for Standardization
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 Lines
SPL	Software product lines
SVN	Subversion
SW	Software

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 http://www.safecer.eu/

[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 (b), 30 September 2017.

[19] AMASS source code https://services.medini.eu/svn/AMASS_source/[3]

[20] AMASS Platform – Prototype Core User Manual https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP1/AMASS_Prototype1_UserManual.docx Version 0.1 (2017)[4]

[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] Richard Hawkins, Software Contribution Safety Argument Pattern (2009) http://www.goalstructuringnotation.info/archives/234

[24] AMASS D3.2 Design of the AMASS tools and methods for architecture-driven assurance (a), 30th June 2017

[25] AMASS D3.4 Prototype for Architecture-Driven Assurance (a), 23 Dec. 2016

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

[27] AMASS D4.3 Design of the AMASS tools and methods for multi-concern assurance (b), planned submission 31 Jan. 2018

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

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

[3] 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 (huascar.espinoza@tecnalia.com)

[4] 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).

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

[31] AMASS D6.5 Prototype for cross/intra-domain reuse (b), December 2017

[32] WEFACT Executable
https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP1/WEFACT.exe

[33] WEFACT user manual and Installation Instructions, 2017
https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP1/WEFACT_UserManual.docx
https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP1/WEFACT_Installation_Instructions.docx

[34] 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.

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

[36] 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.

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

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

[39] 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.

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

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

[42] 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.

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

[44] https://services.medini.eu/svn/AMASS_collab/WP-transversal/ImplementationTeam/PrototypeP1/AMASS_Prototype1_UserManual.docx

Appendix A: Additional CHESS Functionalities

Modelling dependability aspects

As it was documented in D3.2 [24], CHESS implements the conceptual metamodel called SafeConcert [34].

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 [35]), the meta-model used in CHESS toolset to enable component-based systems design.

ConcertoFLA [36] 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. Different FLA techniques are available in the literature [42], 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 Transform Logic (FPTC) [37]. 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 behaviour at system-level, based on the specification of the behaviour of individual components. During the analysis, ConcertoFLA calculates the failure propagation paths and produces their representation according to the specifications of FlaMM meta model (see [38] 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.

Within AMASS, an initial exploration for the exploitation of the failure propagation paths for the generation of FTA was conducted. The work targeted Use Case 11 as a running example and was accepted for publication [39].

In addition, an initial exploration for the exploitation of ConcertoFLA for enabling safety and security analysis was conducted. Based on that exploration, the necessity of extending ConcertoFLA to include failure types for the specialization of dependability threats emerged. The extension however can be done at conceptual level only since ConcertoFLA already includes extension mechanisms and users can add needed failure types. In D4.7 [29], methodological guidelines are provided in order to support the exploitation of ConcertoFLA as it is. In D4.3 [26], a more in-depth exploitation of ConcertoFLA will be taken into consideration, if needed, based on the evaluation of the second iteration of the AMASS prototype.

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 11 - Figure 15) illustrate the usage of the Argument Generator plugin. Further improvements of the generation are under way.

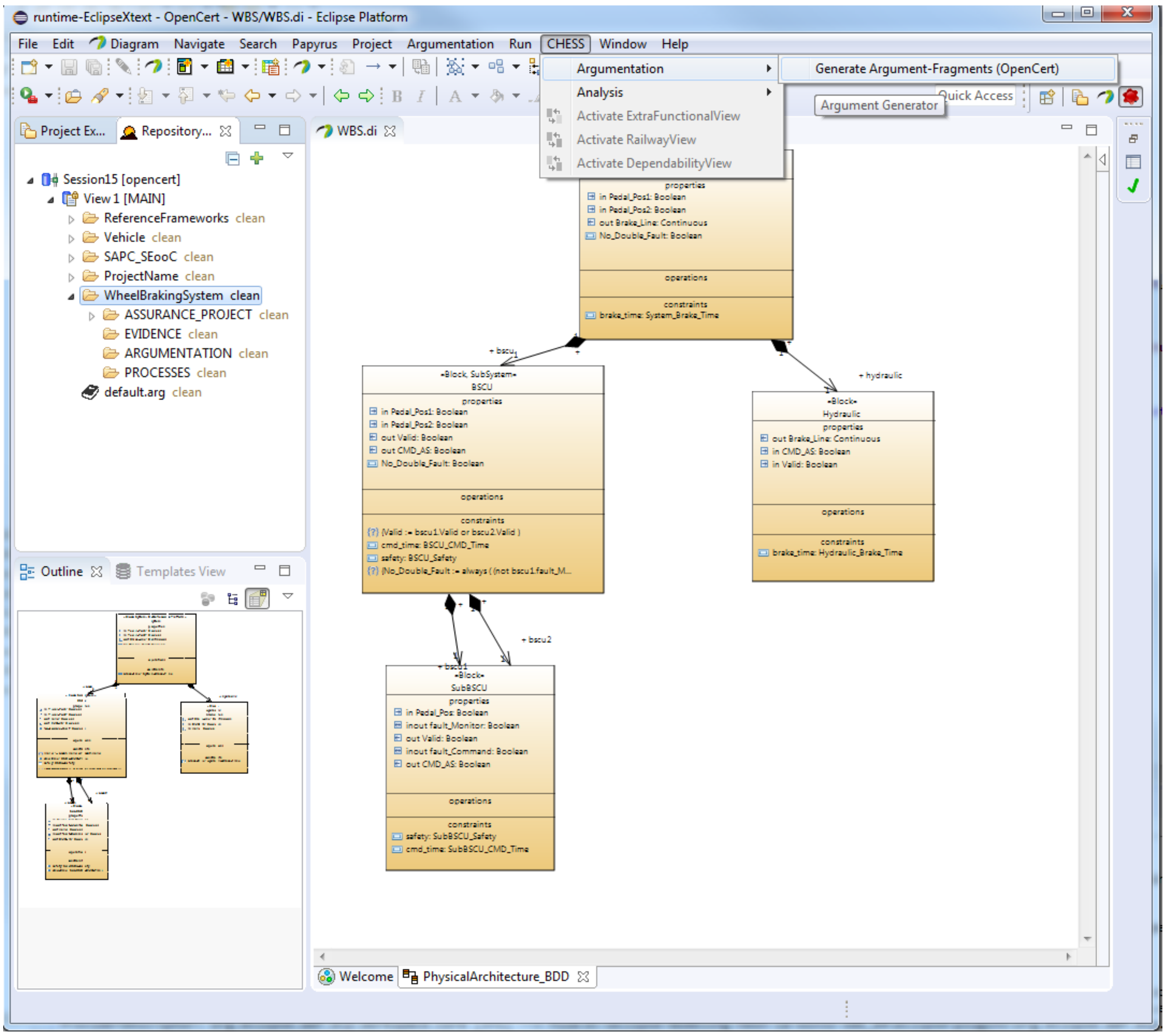
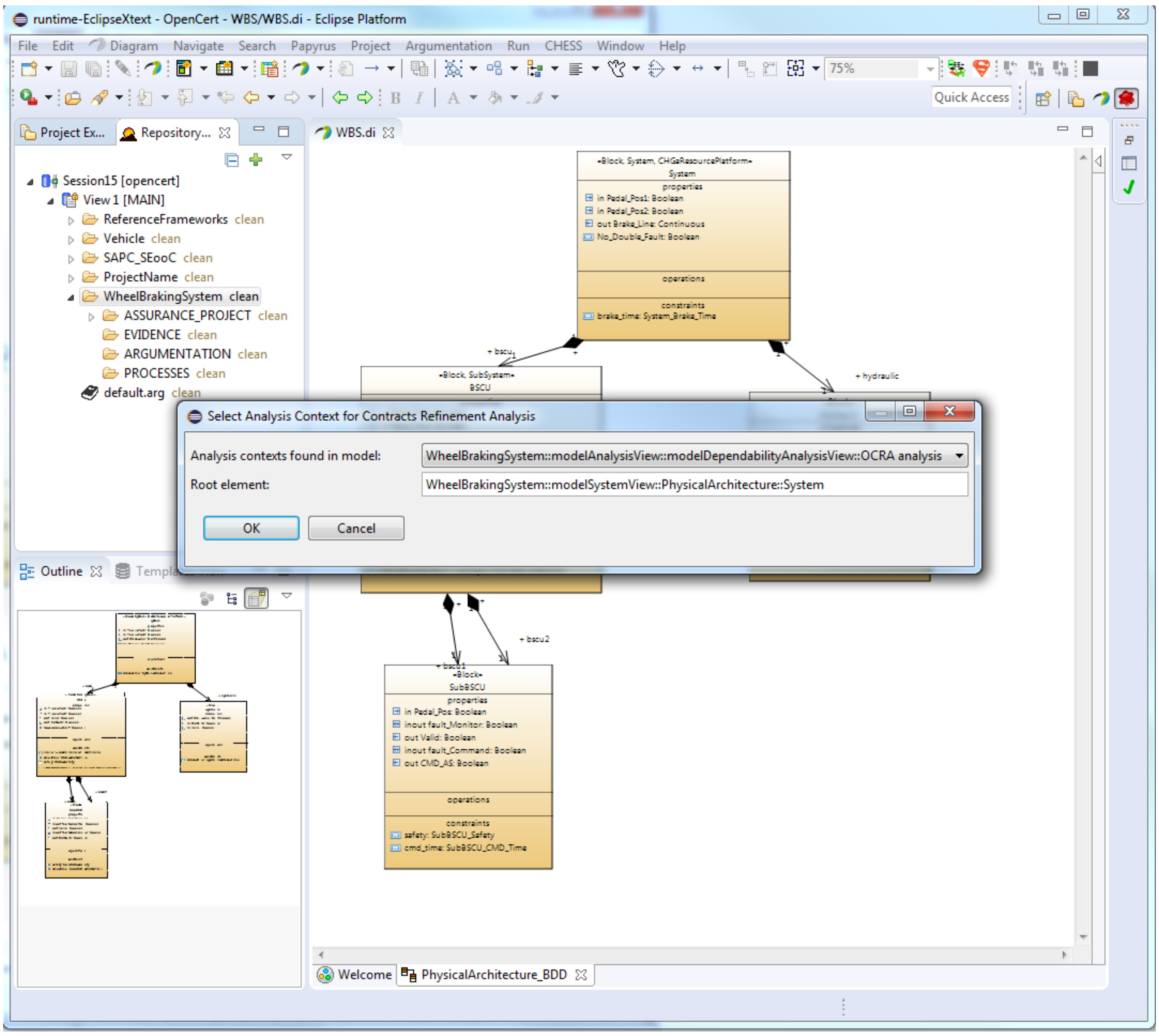


Figure 11. Initiating the argument-fragment generation (Step 1)



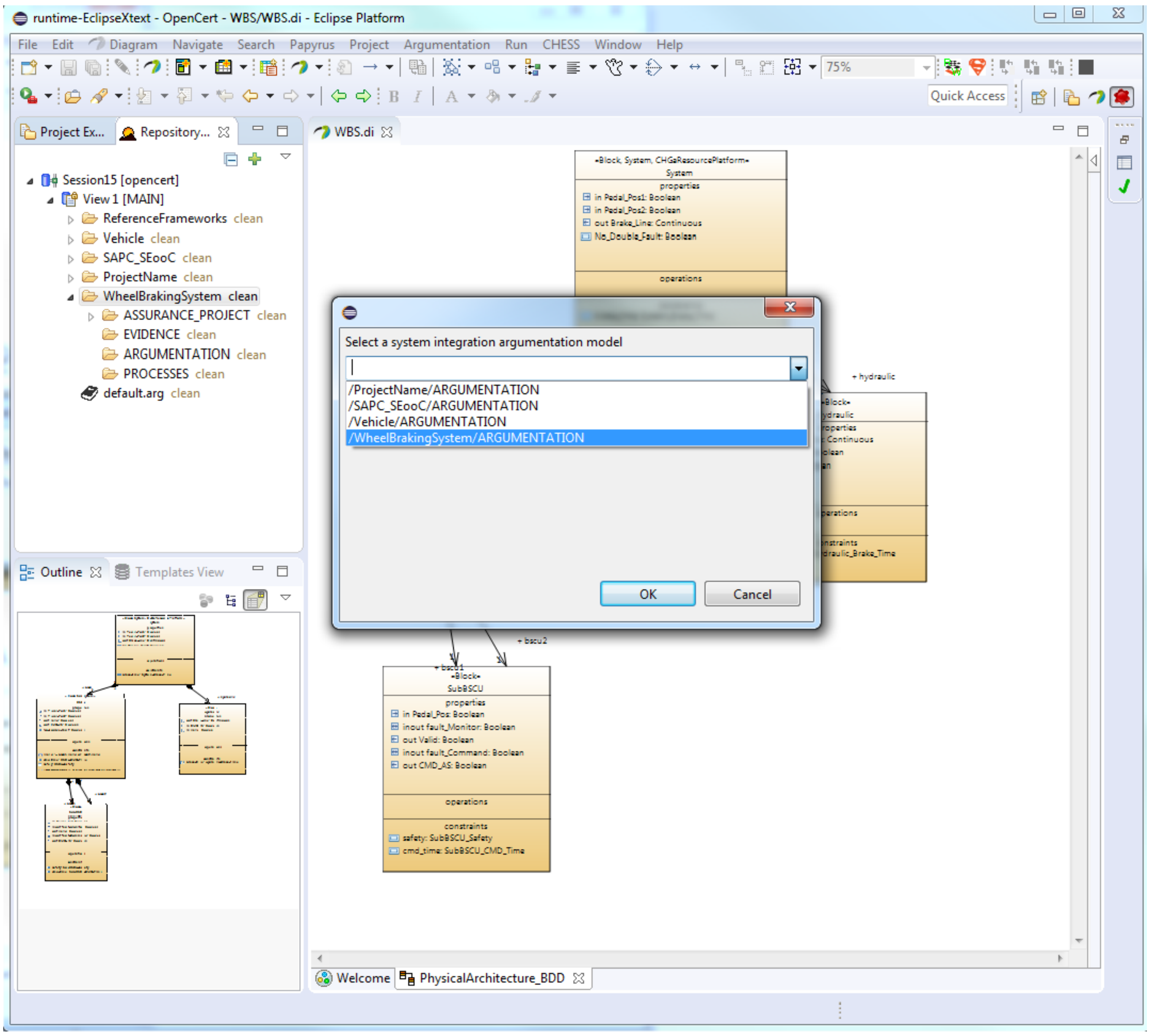
Select Analysis Context for Contracts Refinement Analysis

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

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

OK Cancel

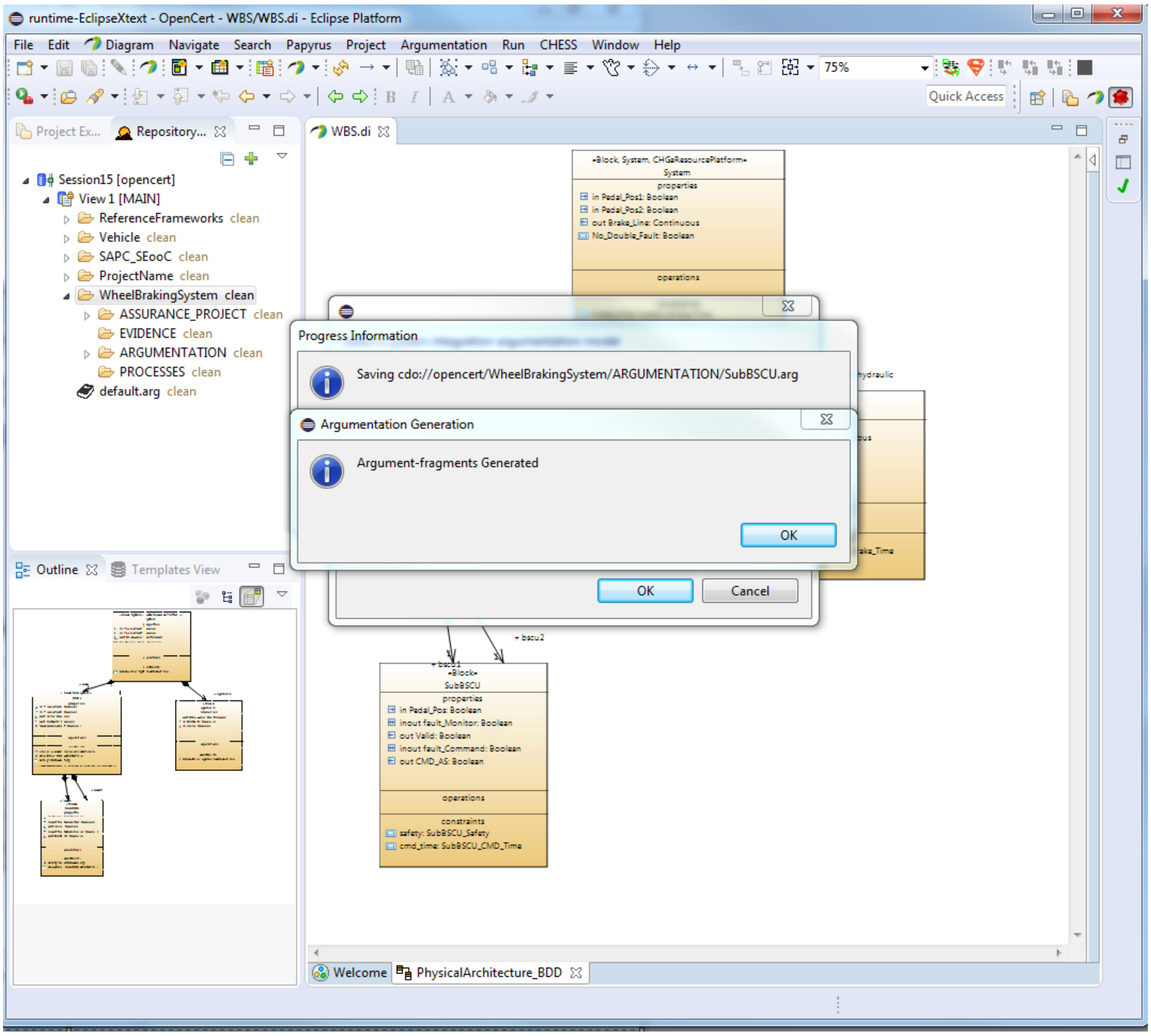
Figure 12. Selecting the source analysis context (Step 2)



Select a system integration argumentation model

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

Figure 13. Selecting the destination assurance case folder on the CDO repository (Step 3)



Progress Information

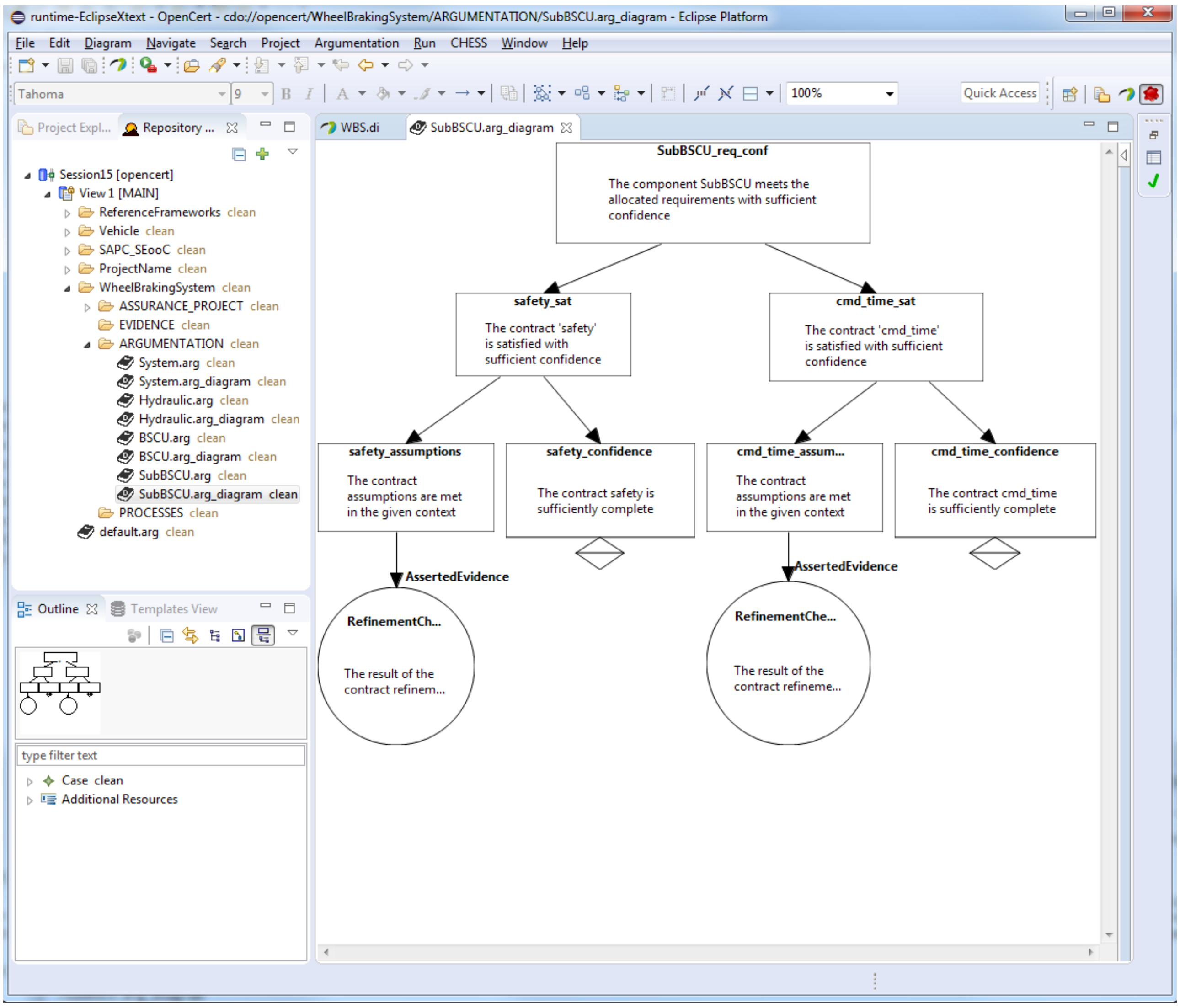
Saving cdo://opencert/WheelBrakingSystem/ARGUMENTATION/SubBSCU.arg

Argumentation Generation

Argument-fragments Generated

OK

Figure 14. Generation successfully completed with argument-fragments for each block



The component SubBSCU meets the allocated requirements with sufficient confidence

The contract 'safety' is satisfied with sufficient confidence

The contract 'cmd_time' is satisfied with sufficient confidence

The contract assumptions are met in the given context

The contract safety is sufficiently complete

The contract assumptions are met in the given context

The contract cmd_time is sufficiently complete

The result of the contract refinem...

The result of the contract refineme...

Figure 15. An example of the generated argument-fragment