ECSEL Research and Innovation actions (RIA)

AMASS

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

Prototype for Architecture-Driven Assurance (a)

D3.4

Work Package:	WP3: Architecture-Driven Assurance
Dissemination level:	PU = Public
Status:	Final
Date:	23 December 2016
Responsible partner:	B&M
Contact information:	Peter M. Kruse <peter.kruse@berner-mattner.com>
Document reference:	AMASS_D3.4_WP3_B&M_V1.0

Contributors

Names	Organisation
Stefano Puri	Intecs
Peter M. Kruse	Berner & Mattner
Huáscar Espinoza, Alejandra Ruiz, Garazi Juez	Tecnalia Research & Innovation

Reviewers

Names	Organisation
Jose María Alvarez (Peer Reviewer)	Universidad Carlos III de Madrid
Tomáš Kratochvíla (Peer Reviewer)	Honeywell
Cristina Martínez, Alejandra Ruiz	Tecnalia Research & Innovation
Jose Luis de la Vara	Universidad Carlos III de Madrid
Barbara Gallina	Maelardalen Hoegskola
Christoph Schmittner	AIT Austrian Institute of Technology GmbH

TABLE OF CONTENTS

List of Figures

Executive Summary

The deliverable D3.4 “Prototype for Architecture-Driven Assurance (a)” is the first output of the AMASS task T3.3 Implementation for Architecture-driven Assurance, whose objective is the development of a tooling framework to support architecture-driven assurance.

AMASS has planned three prototype iterations for the framework; this deliverable reports the status of the aforementioned tooling framework for the first prototype release (aka core prototype), in particular for what regards the system component specification, by describing the supported functionalities and the details about implementation.

This deliverable takes into account the work performed in the other project work-packages, mainly WP2, WP4, WP5 and WP6 because they have strong dependencies with T3.3. Indeed, in this deliverable a set of functionalities regarding the system component specification has been selected from AMASS deliverable D2.1 “Business cases and high-level requirements”. D3.4 describes the technologies that allow the implementation of each selected functionality.

The logical structural view of the AMASS reference tool architecture elaborated in deliverable D2.2 “AMASS Reference Architecture” has been also considered in this deliverable; in particular physical components have been mapped to the logical tool components *Component Editor* and *Contract Editor* identified in deliverable D2.2.

WP4 and WP5 results have been particularly useful for what concerns the argumentation and evidence metamodel specification; indeed one important point related to the implementation for architecture-driven assurance is related to the way system architecture-related information can be traced to the argumentation and evidence models. Two possible solutions are currently under investigation about the implementation of traceability between system architecture and other assurance-related information. These solutions are also presented.

The deliverables D3.5 “Prototype for architecture-driven assurance (b)” and D3.6 “Prototype for architecture-driven assurance (c)” will be the evolution of this deliverable; in particular D3.5 and D3.6 will document the progress about the implementation of the tooling framework supporting architecture-driven assurance.

1. Introduction

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

The AMASS tangible expected 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 (e.g. based on Open Services for Lifecycle Collaboration (OSLC)[3] specifications).

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 Application programming interfaces (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 industrialization. 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[4] is a strong candidate to host AMASS.

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

[1] www.opencoss-project.eu

[2] www.safecer.eu

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

[4] www.polarsys.org

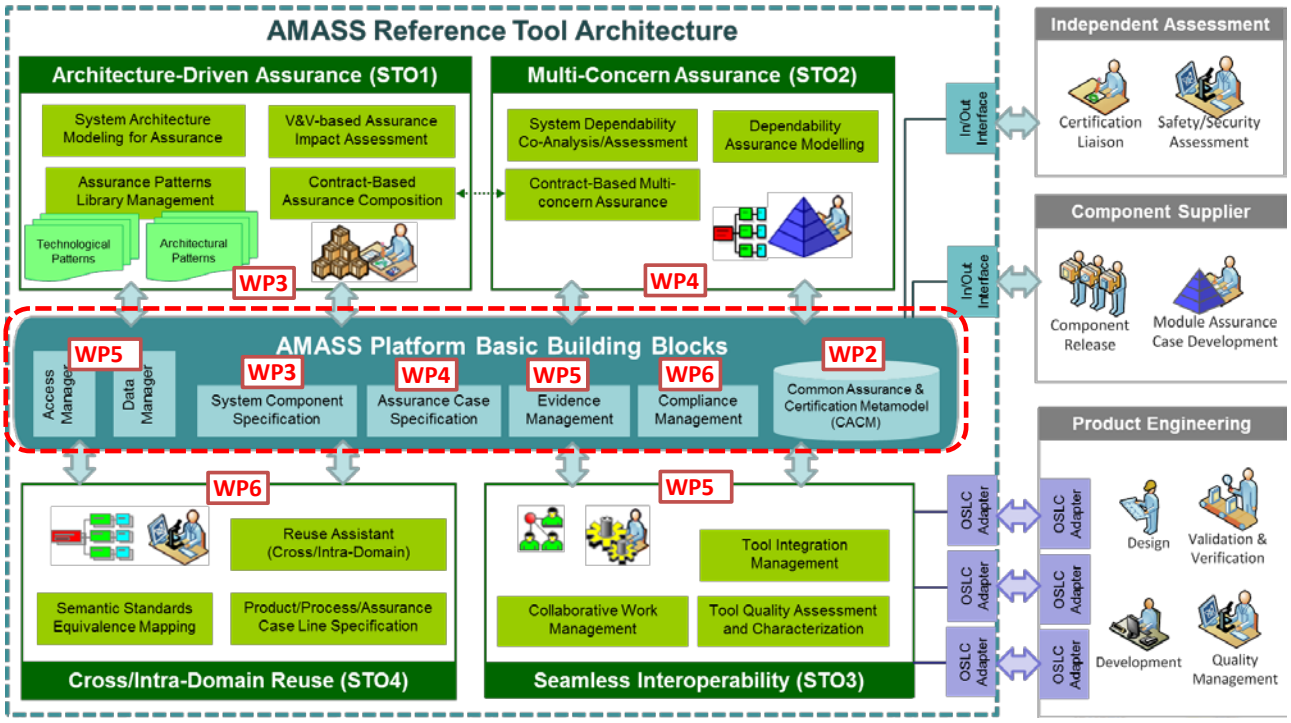

Figure 1: AMASS Building blocks

Since AMASS targets high-risk objectives related to architecture-driven assurance, multi-concern assurance, seamless interoperability support and cross-domain and intra domain assurance reuse, the AMASS Consortium has 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 focusing on a few aspects of the solution.
- Ability to change critical decisions by using practical and industrial feedback (case studies).

AMASS has planned three prototype iterations:

1. During the **first prototyping** iteration (Prototype Core), the AMASS Platform Basic Building Blocks, will be aligned, merged and consolidated at Technology Readiness Level (TRL) 4 (technology validated in laboratory).

2. During the **second prototyping** iteration (Prototype P1), the single AMASS-specific Building Blocks will be developed and benchmarked at TRL 4.

3. Finally, at the **third prototyping** iteration (Prototype P2), all AMASS building blocks will be integrated in a comprehensive toolset operating at TRL 5 (technology validated in relevant environment).

Each of these iterations has the following three prototyping dimensions:

- ***Conceptual/research development***: development of solutions from a conceptual perspective.
- ***Tool development***: development of tools implementing conceptual solutions.

- ***Case study development***: development of industrial case studies using the conceptual and tooling solutions.

As part of the Prototype Core, WP3 is responsible for consolidating the work resulting from previous projects (in particular SafeCer and OPENCOSS, see [9]) on architecture specification in order to design and implement the basic building block called "**System Component Specification**" (see Figure 1). This part of the AMASS platform manages component and contract-based design (see [9] Section 3.1.1).

This deliverable reports the **tool development** results of the "System Component Specification" basic building block. It presents in detail the pieces of functionality implemented in the AMASS platform tools, their software architecture, the technology used, and source code references.

Other important parts of D3.4 document are:

- Installable AMASS Platform tools for the first prototype
- User Manuals and installation Instructions
- Source code description

2. Implemented Functionality

2.1 Scope

The scope for the first prototype iteration (Prototype Core) is the provision of modelling tools for system component specification, including a contract-based approach and the link with the assurance case specification. The main scope is highlighted with a red circle on Figure 2, which shows the general functional overview of the AMASS platform (from AMASS deliverable D2.2 [6]).

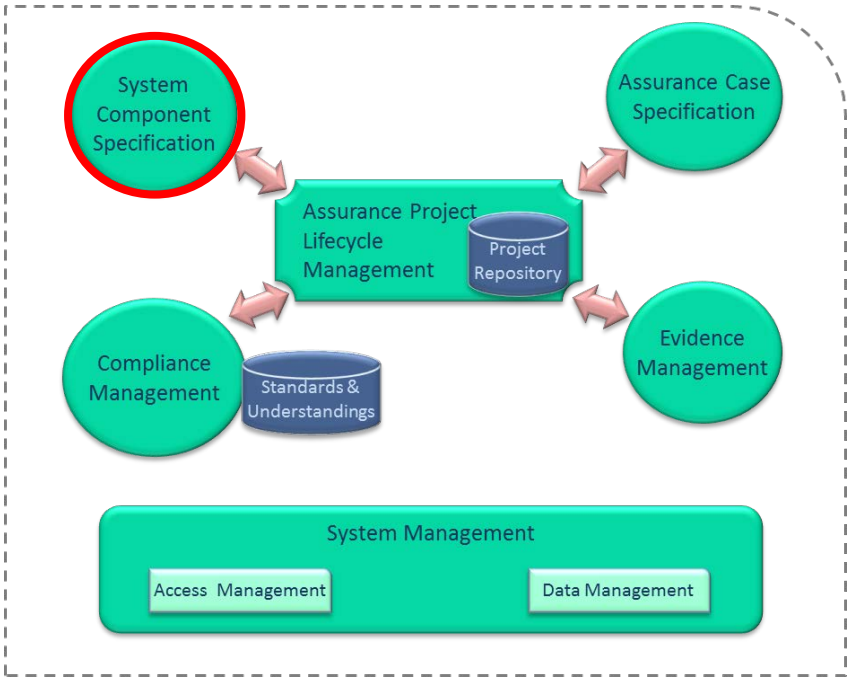

Figure 2: Functional decomposition for the AMASS platform

2.2 Implemented Requirements

From the requirements point of view, this first prototype iteration focuses on a set of AMASS requirements as defined in the AMASS deliverable D2.1 “Business cases and high-level requirements”, currently in progress. Each requirement together with the implementation done so far to implement the requirement is shortly outlined in the following sections.

2.2.1 System Architecture Edition

System architecture specification is supported by the Papyrus UML/SysML editor [5]. The selection of UML/SysML has been driven by the wide adoption of these modelling languages in the industry in different domains. Then, the selection of Papyrus UML/SysML editor has been driven by the fact that Papyrus is one of the most appreciated solid open source tools available in the industry for professional modelling; in particular, recently the Papyrus Industry Consortium has been created to support a model-based engineering platform based on the domain specific and modelling capabilities of the Eclipse Papyrus family

of products. It is worth noting that Papyrus has also integration facilities with other tools, such as the commercial IBM UML Rhapsody tool; in addition, it supports the XMI OMG standard [8] for the interchange of UML models between UML tools.

Through Papyrus editor (see Figure 3), SysML Blocks and UML Components can be used to model the architectural entities as required by the AMASS component meta-model definition (see [6]). Decomposition of block/components into sub-blocks/sub-components can be modelled by using internal block diagrams or composite structure diagrams. Both Papyrus Editor and other AMASS components are under the same Open Source license, which supports the reuse of these previous results into the AMASS platform.

Information about the functional behaviour of a given component/block can be provided through state machine diagrams.

System architecture UML/SysML models and diagrams are stored in individual files in the Eclipse workspace.

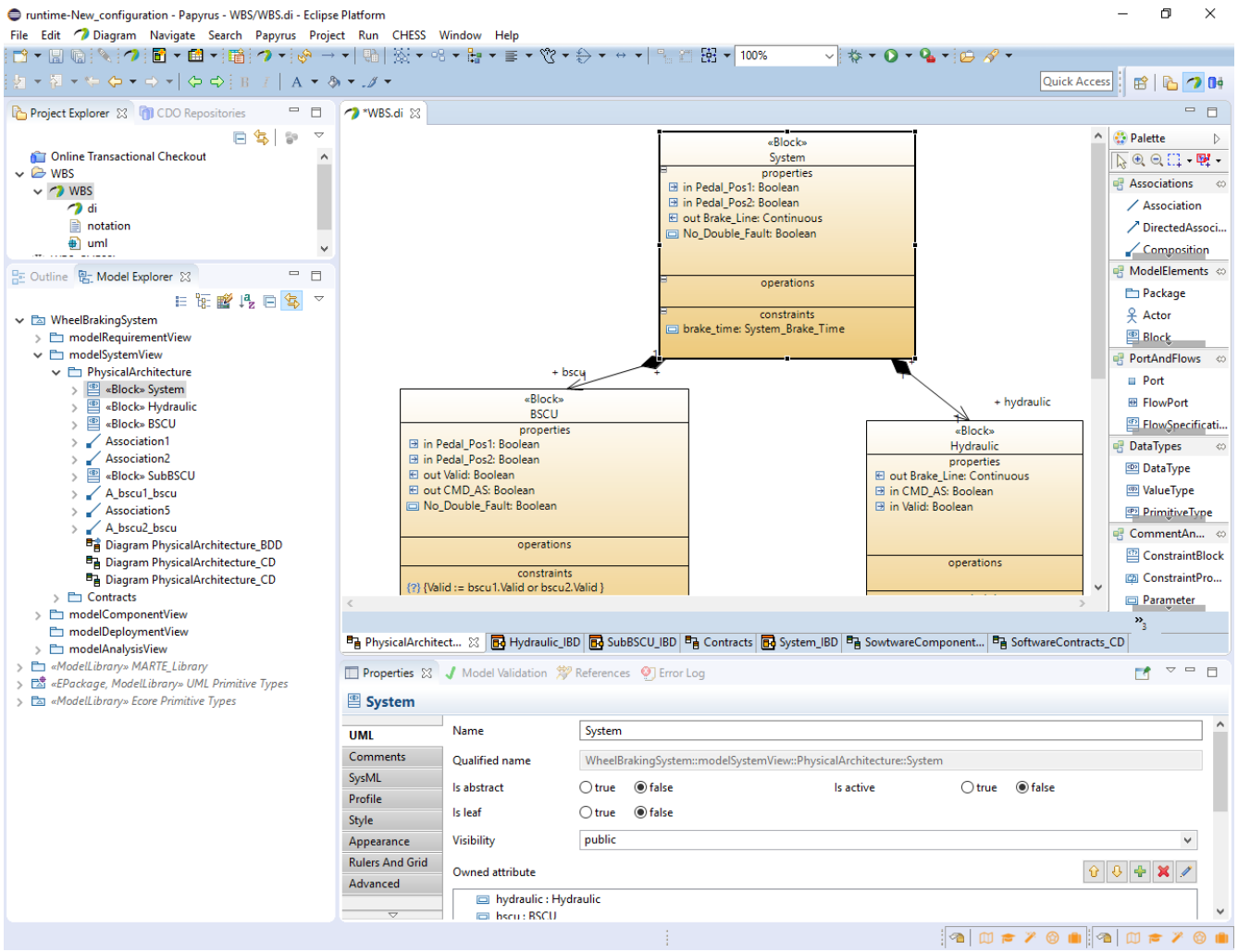

Figure 3: Papyrus editor

The Papyrus UML editor supports the definition and application of UML profiles. In AMASS, Papyrus tool is used together with the CHESS profile extension [3]; in particular CHESS is used here as extension of the

UML and SysML modelling languages to allow the modelling of contracts, as explained in the following sections, according to the AMASS component meta-model needs (see[6]).

CHESS also provides extension to the Papyrus tool, for instance by adding dedicated diagram palettes to facilitate the creation of the CHESS entities, or by adding a dedicated property tabs view for editing CHESS entities properties (see Section 3).

It is worth noting that the CHESS profile also provides other modelling capabilities, such as the dependability profile [10] for failure modelling and specific support for timing properties (see Section 3 about CHESS features). Moreover, CHESS provides a methodology for the design, verification and implementation of CPS SW systems [1]. The aforementioned features are not currently part of the AMASS basic building blocks; their possible role and integration in AMASS will be studied during the project. The CHESS profile follows the same licenses approach as Papyrus and other AMASS components, which supports the easy integration of the developments from the intellectual property perspective.

2.2.2 Formalize Requirements with Formal Properties

Requirements can be modelled in Papyrus using the SysML profile; indeed, SysML comes with the dedicated *Requirement* stereotype (see Figure 4) which can be managed through Requirement Diagrams. The availability of system requirements represented in the model allows to model their traceability to the different parts of the system model. In particular, by using the SysML profile, Requirements can be traced to the entities of the architecture, by using the *Satisfy* link defined by SysML. In this way, model-driven support can be enabled to support requirement traceability (see e.g. [7]), which is an important quality factor to be guaranteed while building systems.

In AMASS, a formal property represents a distinct entity which is used to provide a formal description of a given system requirement, the latter usually described using informal textual language.

To model formal properties, CHESS profile defines a class called **`FormalProperty`** as an extension of UML Constraint (see Figure 4). A `FormalProperty` can be created first in the model and then it has to be linked to the requirement that it formalizes; the SysML trace link can be created in the SysML Requirement diagram or through the tabular editor provided by Papyrus[5]. Then the formal description of the requirement has to be provided by using the *specification* attribute coming with the `FormalProperty` entity.

[5] https://wiki.eclipse.org/Papyrus_User_Guide/Table_Documentation

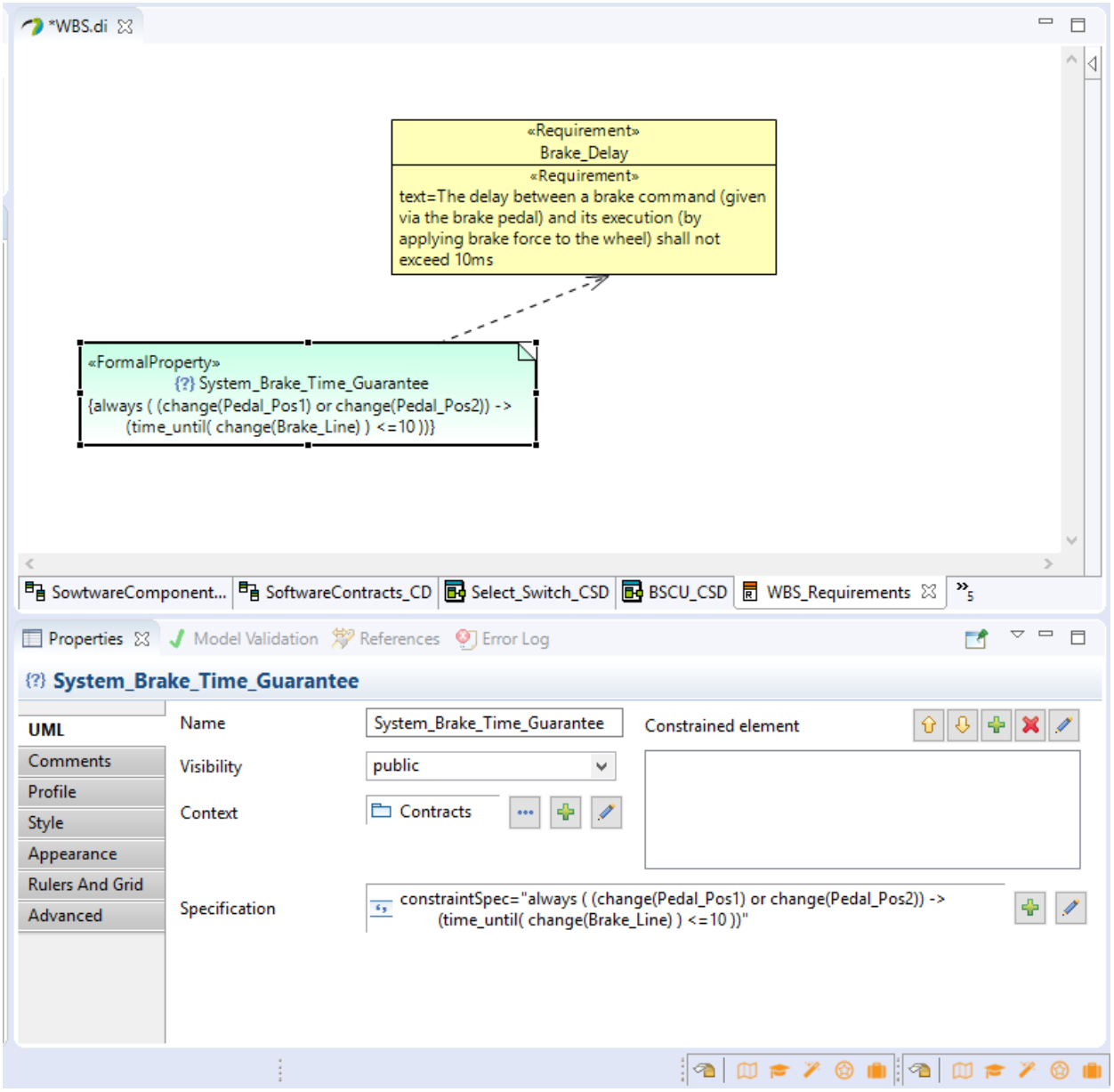

Figure 4: Modelling FormalProperty

It is worth noting here that the CHESS profile does not force the usage of a particular formal language; the choice of the formal language to be adopted for the formalization of requirements is made by the modeler, typically according to the adopted process/methodology. CHESS currently supports integration with the OCRA contract specification language[6]; in particular, through the CHESS Contract plugins explained in Section 3 it is possible to verify formal properties specification with respect to OCRA syntax.

[6] https://ocra.fbk.eu

2.2.3 Structure Properties into Contracts

CHESS profile supports the modelling of weak and strong contracts to support contract-based design (the reader can refer to AMASS D3.1 [9] for an introduction to weak and strong contracts and contract-based design).

Contracts are available in the CHESS profile as a special kind of classifiers (i.e. an entity used to describe instance-level entities of the same kind). Contracts can be created in UML class, component, or SysML block diagrams. A Contract comes with two attributes representing the assumption and guarantee formal properties.

By using the CHESS Papyrus extension, when a Contract is created in the model, the tool automatically creates a pair of empty `FormalProperties`, the latter playing the role of assumption and guarantee of the Contract itself.

Alternatively, a given `FormalProperty` available in the model before the creation of the Contract can later be assigned to the Contract itself, as assumption or guarantee.

Figure 5 below shows an example of `Contract` and `FormalProperty` modelling; the figure shows the `Assume` and `Guarantee` attributes owned by the `Contract`, which in the example are bounded to the represented `FormalProperty`. A link between the `Contract` and the `FormalProperty` is also depicted.

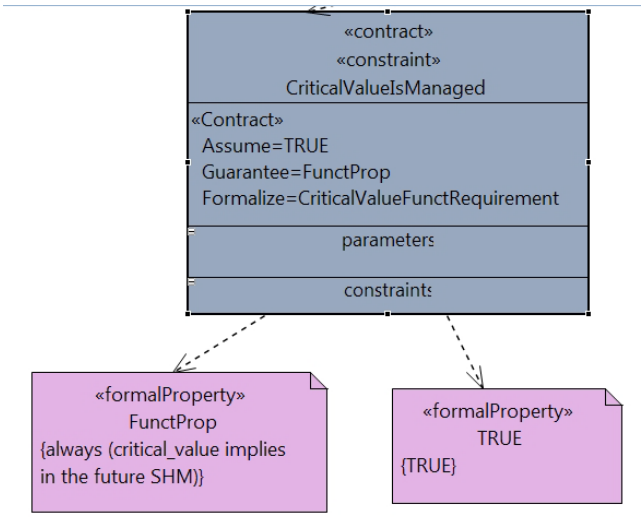

Figure 5: Contract and FormalProperty example

2.2.4 Assign Contract to Component

In CHESS, a Contract is assigned to a given UML Component/SysML Block by instantiating the Contract itself in the Component/Block. In particular, a *`ContractProperty`* attribute has to be created for the Component/Block first and then the *`ContractProperty`* type must be set to the particular *`Contract`*. Therefore, in CHESS one important piece of information related to contract-based design is modelled through the *contract instance*, which represents a *`Contract`* associated to a Component/Block.

ContractProperty has also an attribute that allows to specify if the associated Contract has to be applied to the Component/Block according to the weak or strong semantics[7] [9].

As example, Figure 6 shows the *criticalValueIsManaged ContractProperty* owned by the *FunctionalSystem* Block (the *ContractProperty* is shown in the diagram in the *Constraint* compartment of the Block). The *criticalValueIsManaged* property is typed as *CriticalValueIsManaged* Contract (*criticalValueIsManaged:CriticalValueIsManaged*), the latter also represented in the diagram. The *criticalValueIsManaged* property represents the association of the *CriticalValueIsManaged* Contract *to the FunctionalSystem* Block.

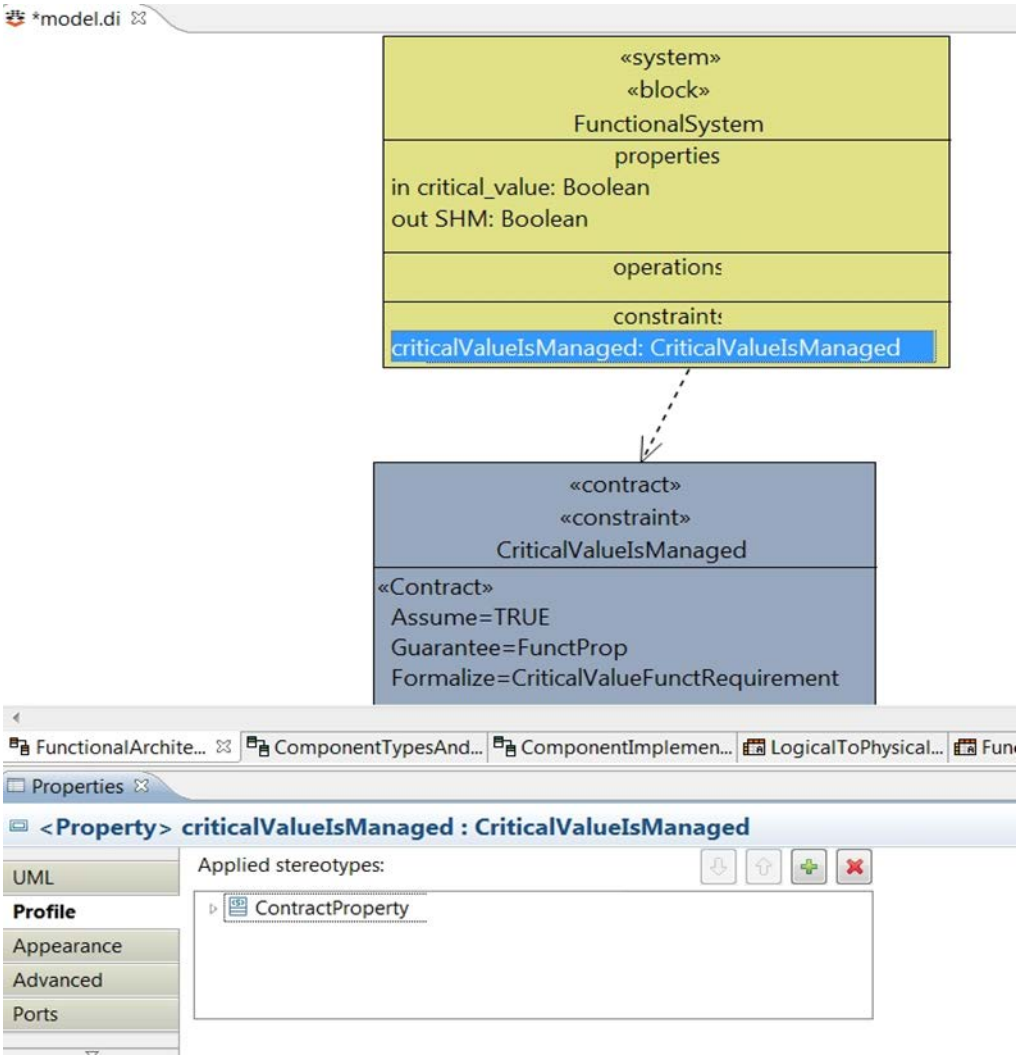

Figure 6: Assign Contract to Component

[7] As discussed [9], while strong assumptions define compatible environments in which the component/block can be used, weak assumptions define specific contexts where additional information is available. Hence, a component/block should never be used in a context where some strong assumptions are violated, but if some weak assumptions do not hold, it just means that the corresponding guarantees cannot be relied on.

This allows to potentially reusing the same *Contract* in different contexts/systems (as analogous to the practice of sharing requirements across projects, i.e. software/system requirements reuse).

2.2.5 Contract Refinement

The CHESS profile allows to model contracts refinement/decomposition along the refinement/decomposition of the architectural entities, the latter provided through UML composite structural diagrams or SysML block definition diagrams. In particular, contract instances play a key role during the refinement specification. Indeed, contracts refinement is modelled for contract instances, not for the Contracts entities; this is because the same Contract can be reused in several contexts (i.e. instantiated in several Components/Blocks), and for each context the refinement of the same Contract could be different. So through the CHESS profile it is possible to model how a given contract instance is refined by a set of other contract instances.

In practice, given a contract instance C assigned to a component A, and given the decomposition of A into subcomponents (A_1,…,A_n) and the contracts instances assigned to each subcomponent ($C_{1<1..k>}$,… ,$C_{n<1..j>}$)., it is possible to model how C is decomposed by (a subset of) ($C_{1<1..k>}$,…,$C_{n<1..j>}$).

2.2.6 Modelling Failure Behavior

For the modelling of failure and security behavior (e.g. an accidental / malicious fault occurring at a given component's input/output port) no specific implementation has been currently identified as official part of the AMASS building block; this is currently an ongoing task in the AMASS project.

Existing support for failure behavior modelling is available from state of the art projects and modelling tools, like the UML/MARTE dependability profile coming with the CHESS modelling language [3].

Explicit support about the modelling of failure behaviour, and in particular about any implementation about its role in the context of the architecture-drive assurance objective, will be provided in the next release of this deliverable.

2.2.7 Link Architecture-Related Entity to Assurance Case Information

The allowed links between architectural entities and the other parts of the CACM AMASS meta-model (about management of the assurance project as indicated in Figure 2) are currently described in AMASS deliverable D2.2 [4].

As explained in the previous sections, the AMASS component model has been made available in Eclipse as UML/SysML extended with the CHESS profile for contracts, while the other parts of the CACM (argumentation, evidence, compliance management) are currently implemented as Ecore meta-mode[8] (not as UML profile).

Within the UML profile definition, it is not possible to refer to an Ecore entity which is not related to the UML language, so the aforementioned links (e.g. from a CHESS-Contract to an argumentation-Claim) cannot

[8] Ecore is a model provided by the Eclipse EMF project (https://www.eclipse.org/modeling/emf); Ecore can be used to model the structure of a given domain of data models. Typically, Ecore is referenced as meta-meta-model; the structure of a given domain of data models is referenced as meta-model, where a model is a concrete instance of this meta-model.

be expressed through the CHESS profile; the links have to be managed with some additional modelling support, as explained below in the text.

Indeed, in the context of Task 3.3 we are currently investigating the best approach to allow the modelling of the links between the component model entities and the other parts of the CACM. One solution could be to use the **EAnnotation** mechanism available in Ecore: EAnnotation allows to attach extra information to any object available in an Ecore model. In our case, EAnnotation could be created for a UML model entity (for instance a Contract)[9]; then EAnnotation could be used to refer to an entity of the CACM defined in some external (to the UML) model (as a Claim in an argumentation model). Figure 7 gives a picture of what has been stated above (CACM model in the figure has to be intended as the model for argumentation, evidence, and compliance management).

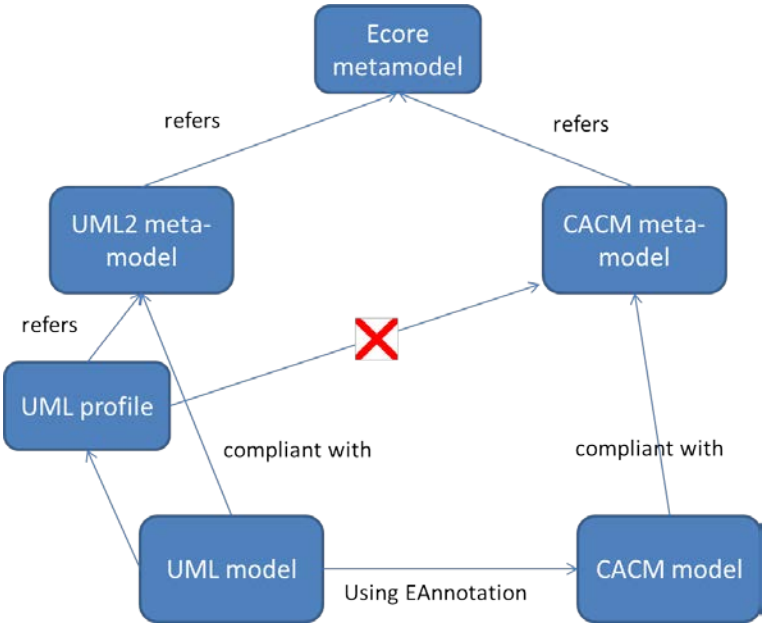

Figure 7: Links through EAnnotation

Another possibility is to use a traceability support based upon a dedicated traceability meta-model (see Figure 8). In this way, a link would be created according to the traceability meta-model; each link would own a reference to the UML model entity and a reference to the CACM model entity to be associated. We are also currently evaluating the applicability of this approach to the needs of traceability addressed in WP5.

[9] It is worth noting that EAnnotation can be added to UML model entities because UML models in Eclipse are implemented as Ecore models.

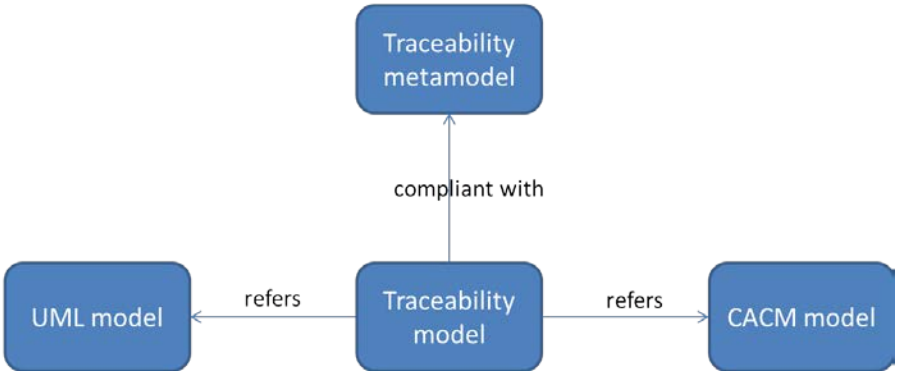

Figure 8: Links through traceability meta-model

It is worth noting that both the aforementioned solutions can also be used to model links between architectural entities and process related information, being the latter defined according to WP6 results (at the time of writing, the specification of the links between architectural entities and process related information has not yet been fully formalized, it will be defined in the context of task T3.2).

We are still investigating the benefits and limits of each of the aforementioned solutions, to finally select the one to be implemented. What is worth noting is that the usage of a dedicated traceability meta-model could be made generic in order to support traceability between assurance case information and architecture-related entities specified with other non-UML modelling languages. For instance, by assuming the availability of an Architecture Analysis and Design Language (AADL)[10] editor in Eclipse, the same traceability model could be used to create links between AADL entities and argumentation/evidence entities available in the CACM model.

2.3 Installation and User Manuals

The steps necessary to install the first prototype are going to be exhaustively described in the AMASS User Manual (currently in progress) and will not be repeated here. That document will contain all required steps and document references to set up the tools. There is currently no pre-packaged distribution.

In summary, this document is a user manual of the first AMASS tool prototype implementation. The users can find the installation instructions, the tool environment description, and the functionalities for the creation of Standards and Process models (models representing Standards, Regulations, or Company-specific Processes), Assurance Projects and the associated Evidence models (Artefacts), Compliance Maps (so far, compliance maps from Reference Artefacts to Artefacts), and Argumentation models, in addition to Architecture models.

[10] http://www.aadl.info/aadl/currentsite

3. Implementation Description

3.1 Implemented Modules

As documented in AMASS deliverable D2.2 [6], the System Component Specification logical building block decomposes into two sub-blocks (see Figure 9): the Component Editor and the Contract Editor. The purpose of the first tool module is to provide services for architecture specification; the second tool module provides services to store and instantiate contracts and to associate them to the architectural entities.

The two aforementioned blocks and associated services are made available in the AMASS platform through the usage of the Eclipse-Based Papyrus UML/SysML Editor extended with the CHESS plugins. In particular, Papyrus contains plugins for edition of architectural/component-based models, together with the possibility to model requirements (by using the SysML profile support). CHESS provides plugins for management of formal properties and contracts specification and their association to the architectural components.

The CHESS profile for Contract (see D3.1) is implemented as a UML/SysML profile; the profile has been designed using the Papyrus editor facilities.

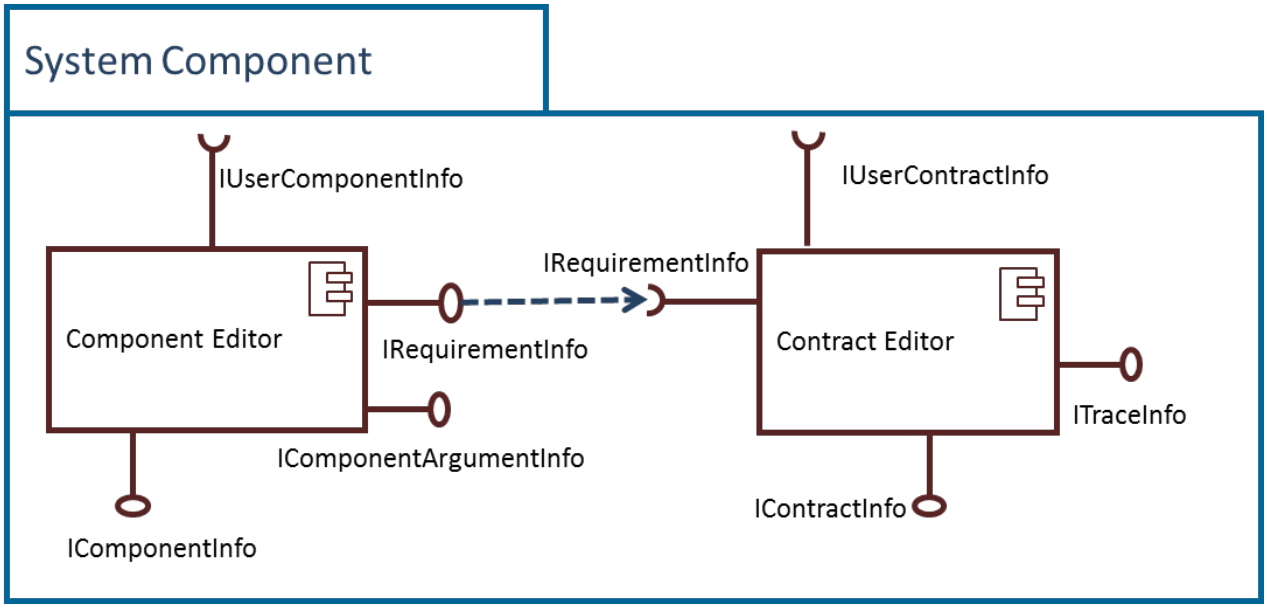

Figure 9: Tool modules for System Component Specification

3.2 Source Code Description

Papyrus[11] is an Eclipse project and its source code is freely available through the Eclipse GIT server[12].

[11] https://eclipse.org/papyrus/

The source code of the CHESS contract editor is available through the Polarsys CHESS project[13].

Extensions to the Polarsys CHESS project are foreseen during the context of AMASS project; the extensions will be developed by working on an AMASS dedicated code repository (https://services.medini.eu/svn/AMASS_source). Then, once the extensions are sufficiently mature, they will be pushed to the Polarsys CHESS repository as AMASS contribution.

The additional CHESS plugins that need to be installed on top of Papyrus environment to enable the CHESS-based AMASS Contract Editor features are the following (see also Figure 10):

- `org.polarsys.chess.contracts.chessextension`: provides the Papyrus extension to easily work with the CHESS Contract profile, for instance to facilitate the creation of CHESS stereotypes.
- `org.polarsys.chess.contract.integration`: implements the integration with the OCRA and XSAP tools; in particular, it allows automatically invoking the aforementioned tools and getting back the obtained results within the Eclipse environment.
- `org.polarsys.chess.contracts.profile`: implements the CHESS profile for contracts.
- `org.polarsys.chess.contracts.transformations`: implements the model of text transformation for the integration with the OCRA and XSAP tools; in particular a corresponding OCRA model can be generated starting from the components and contracts modelled in UML/SysML and CHESS profile. The plugin adds dedicated command to the CHESS Eclipse menu to invoke the transformations.
- `org.polarsys.chess.contracts.validation`: implements the validation of the constraints that the CHESS model has to satisfy in order to allow the mapping to the OCRA language and then the integration with the OCRA tool.
- `org.polarsys.chess.contracts.feature`: allows to deploy/undeploy the CHESS plugins related to contract-based design support.

12 https://git.eclipse.org/c/papyrus/org.eclipse.papyrus.git/

13 https://git.polarsys.org/c/chess/chess.git?h=develop

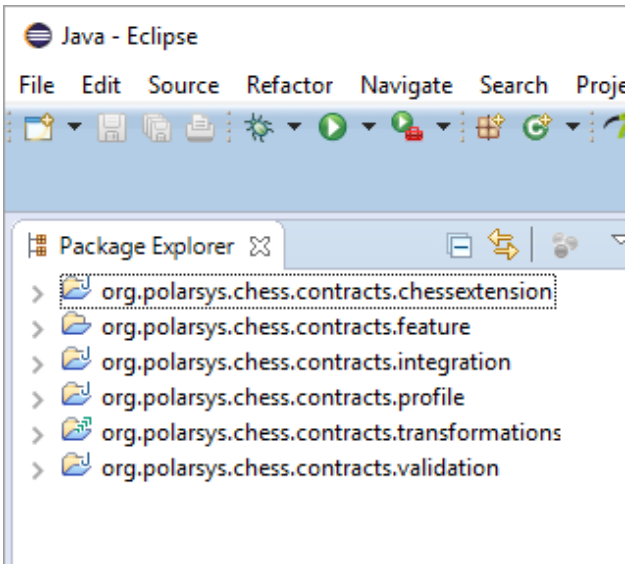

Figure 10: CHESS plugins supporting Contract Based Design

One important point to mention is that, in addition to the aforementioned support for contract design, the Polarsys CHESS project provides additional features.

In particular, the Polarsys CHESS project provides a set of *core* plugins that allow the application of the CHESS methodology ([1][2]). This is the base upon which the AMASS methodology will build. The actual CHESS methodology allows the design, verification and implementation of cyber physical software systems; CHESS adopts a dedicated component model language [4] and ad-hoc model transformations to enable timing/dependability analysis and code generation. Moreover, the CHESS methodology defines a multi-view approach for modelling the different aspects/concerns of the system; for each view, the diagrams and entities that can be created/viewed/modified are fixed and formalized in the view definition. The CHESS plugins extend the Papyrus editor to support the CHESS modelling language and design-by-view approach; so, by using the CHESS Papyrus extension, the constraints imposed by the CHESS methodology are enforced in a live-manner, at modelling time, to avoid late discovery of modelling activities which can violate the correctness-by-construction approach implemented by CHESS.

The CHESS-based AMASS Contract plugins use some utilities provided by other core CHESS plugins; in detail, the core CHESS plugins used are:

- `org.polarsys.chess.core`: provides some facilities regarding selections and diagram status.
- `org.polarsys.chess.services`: provides functionalities about the CHESS editor (as extension of the Papyrus one).
- `org.polarsys.chess.validation`: provides functionalities about model validation.
- `org.polarsys.chessmlprofile`: provides the SysML/UML/MARTE profile implementation of the CHESS modelling language [3]. Moreover, it provides dedicated diagram palettes extending the Papyrus ones to easily manage the creation of CHESS stereotypes in a given diagram.

Therefore, CHESS core plugins are required in order to use the CHESS Contract feature.

In order to allow the AMASS platform's stakeholders to use the CHESS-based AMASS Contract features on top of the Papyrus editor without having to use the CHESS methodology for SW development, an extension

has been made to the CHESS core plugins. In particular, the user can decide to disable the live-check of the constraints associated to the CHESS multi-views support; in this way, the modeler can use the full Papyrus and UML features, together with the CHESS extension for contract based design.

Figure 11 below provides a snapshot of the CHESS methodology constraints that can be enabled/disabled through the Eclipse preferences page.

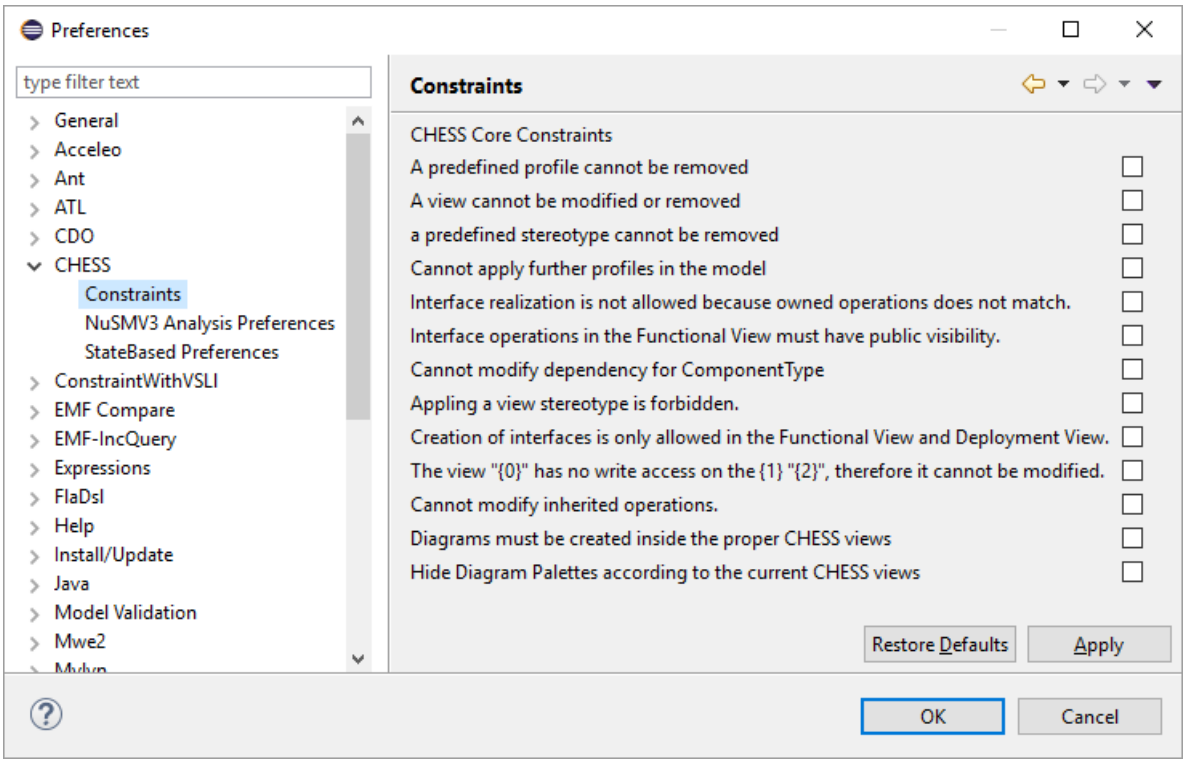

Figure 11: CHESS methodology constraints

4. Abbreviations

Abbreviation	*Explanation*
AADL	Architecture Analysis and Design Language
API	Application Programming Interface
CACM	Common Assurance and Certification Meta-model
CHESSML	CHESS Modelling Language
CPS	Cyber Physical System
ECSEL	Electronic Components and Systems for European Leadership
EMF	Eclipse Modeling Framework
JU	Joint Undertaking
MARTE	Modeling and Analysis of Real Time and Embedded systems
OCRA	Othello Contracts Refinement Analysis
OMG	Object Management Group
OSLC	Open Services for Lifecycle Collaboration
SW	Software
SysML	System Modeling Language
TRL	Technology Readiness Level
UML	Unified Modelling Language
V&V	Verification and Validation
WP	Work Package
XMI	XML Metadata Interchange
XSAP	eXtended Safety Assessment Platform

5. References

[1] Mazzini S., J. Favaro, S. Puri, L. Baracchi., "CHESS: an open source methodology and toolset for the development of critical systems", 2nd International Workshop on Open Source Software for Model Driven Engineering (OSS4MDE), Saint-Malo, October 2016

[2] L.Baracchi, S.Mazzini, S.Puri, T.Vardanega: "Lessons Learned in a Journey Toward Correct-by-Construction Model-Based Development", Reliable Software Technologies – Ada-Europe 2016 Volume 9695 of the series Lecture Notes in Computer Science pp 113-128, 31 May 2016

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

[4] CONCERTO ARTEMIS JU project, D2.2 The CONCERTO Component Model, 9 May 2014, available at http://www.concerto-project.org/results

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

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

[7] M. dos Santos Soares, J. Vrancken: "Model-Driven User Requirements Specification using SysML", JOURNAL OF SOFTWARE, VOL. 3, No. 6, June 2008

[8] XML Metadata Interchange, www.omg.org/spec/XMI/

[9] AMASS D3.1 Baseline and requirements for architecture-driven assurance, 30 September 2016

[10] CONCERTO D3.3 – Design and implementation of analysis methods for non-functional properties - Final version, 18 November 2015, Public Distribution, http://www.concerto-project.org/results