OETCS/WP7/07.3.5

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

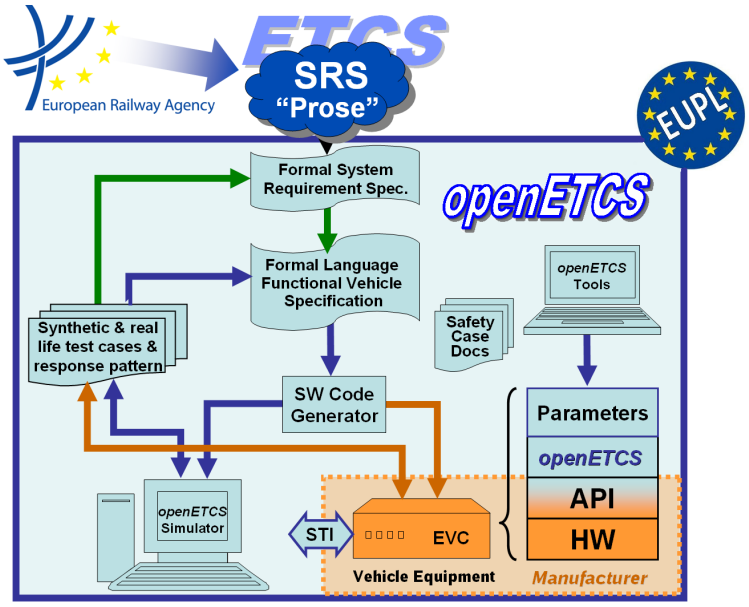
Work-Package 7: “Toolchain”

Traceability Architecture in OpenETCS

WP7 Proposition

Cecile Braunstein, Moritz Dorka, David Mentré and Raphaël Faudou

November 2015



Funded by:

Federal Ministry of Education and Research
Région de Bruxelles-Capitale
GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, ENERGÍA Y TURISMO

This page is intentionally left blank

Work-Package 7: “Toolchain”

OETCS/WP7/07.3.5
November 2015

Traceability Architecture in OpenETCS

WP7 Proposition

Document approbation

Lead author:	Technical assessor:	Quality assessor:	Project lead:
location / date	location / date	location / date	location / date
signature Cécile Braunstein (University Bremen)	signature ()	signature ()	signature ()

Cecile Braunstein

University Bremen

Moritz Dorka

DB

David Mentré

Mitsubishi Electric R&D Centre Europe

Raphaël Faudou

Samares Engineering on behalf of ENSEEIHT

OpenETCS : Position Paper on traceability

Prepared for openETCS@ITEA2 Project

Abstract: This document presents a proposition to the tool chain traceability architecture.

Table of Contents

Document Information

Document information	
Work Package	WP7
Deliverable ID or doc. ref.	O7.3.5
Document title	Traceability Architecture in OpenETCS
Document version	00.02
Document authors (org.)	Cécile Braunstein (Uni.Bremen)

Review information	
Last version reviewed	
Main reviewers	

Approbation			
	Name	Role	Date
Written by	Cécile Braunstein	WP7-T7.3 Sub-Task Leader	06.02.2014
Approved by			

Document evolution			
Version	Date	Author(s)	Justification
00.00	17.12.2014	C. Braunstein	Document creation
00.00	23.10.2015	R. Faudou	Precisions concerning OpenETCS requirements and models and update of tool chain traceability requirements

1 Introduction

1.1 Document purpose

This document presents a proposition concerning support of openETCS project traceability activity by openETCS tool chain. This proposition is a consensus between traceability needs, project policy (open source solutions), time and efforts (focus on existing tools and features) and risks about tool maturity.

1.2 Document scope

This proposition is based on the needs and priorities about traceability, captured from different work packages (mainly WP3 and WP4) during October 2015, either from interviews or from reading of available project documents, especially:

- Project Quality Assurance Plan - D1.3 - [1].
- Definition of OpenETCS Development Process - D2.3a v02 - [2]
- OpenETCS Architecture and Design Specification - D3.5.0 - [3] and D3.5.1 - [4]
- Safety plan - D4.2.3 - [5]

The proposition done in this document is based on existing tools and existing tool capabilities with verified maturity. Document will explain limits of current proposition and will suggest axes to investigate in the future of project (roadmap). But detailed evaluations of tools and features based on those investigations is not part of this document.

1.3 Document organization

Chapter 2 recalls openETCS project scope, development process, different needs and requirements to address, architecture and design specification.

Chapter 3 highlights main priorities concerning traceability and detailed expectations about traceability activities to support (including design, code, verification and validation).

Chapter 4 defines tool chain requirements concerning traceability.

Chapter 5 presents current main realistic traceability architecture to support tool chain requirements with existing mature tools.

Chapter 6 summarizes limits of the proposition and suggests axes to investigate in order to improve traceability support by the OpenETCS tool chain.

2 OpenETCS traceability scope

Requirement traceability concerns relations between requirements existing at different engineering levels and relations between requirements and other engineering artifacts (models, documents, test cases, code...). All those requirement traceability links can have different semantics including derivation, refinement, satisfaction, implementation and verification.

Before defining traceability process and the different link types, it is important to clearly define the scope of openETCS activity and the requirements that we want to trace. Next paragraphs recall OpenETCS vision, design process and VV process.

2.1 Recall of OpenETCS target vision and development process

[2] mentions that openETCS activity pursues the vision of a full CENELEC compliant development of open-source software for the *European Vital Computer* (EVC). It is intended to produce a software kernel which can be used in commercial EVCs.

CENELEC 50128:2011 is the relevant engineering standard for software development of safety-critical rail systems. It is completed with CENELEC 50126-1:1999 to address system and subsystem engineering activities required to prepare software development with system requirements.

[5] recalls that *the main products of the openETCS project will be the openETCS specification model used to generate the openETCS on-board software and the openETCS tool chain development, which is used to formalize the ERA Specifications for ETCS, generate the Software Code and perform verification and validation activities*

OpenETCS software development process reflects this model-based approach while remaining compliant with CENELEC engineering standards.

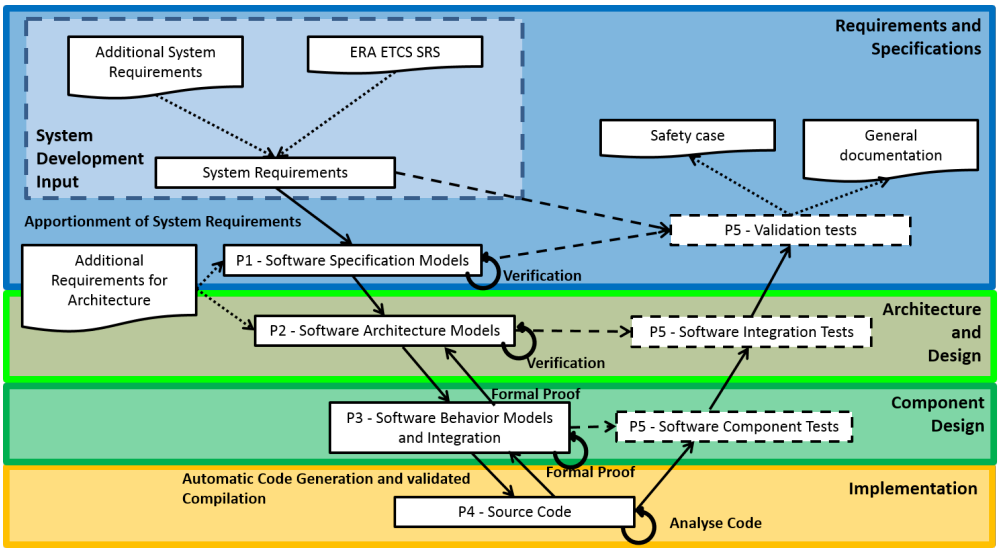


Figure 1. OpenETCS software development process

2.2 OpenETCS project context-of-interest

[4] recalls last refinement of project scope concerning functional coverage: "OpenETCS project has decided not to consider all ETCS subsystems and to tailor scope to cover the functionality required for the openETCS demonstration as an objective of the ITEA2 project. The goal is to develop a formal model and to demonstrate the functionality during a proof of concept on the ETCS Level 2 Utrecht Amsterdam track with real scenarios".

Figure 2 details ETCS system scope to consider in order to cover expected demonstration of EVC software for ITEA project. Scope is highlighted in ERA TSI chapter 2 and provides basis for System detailed analysis and functional breakdown.

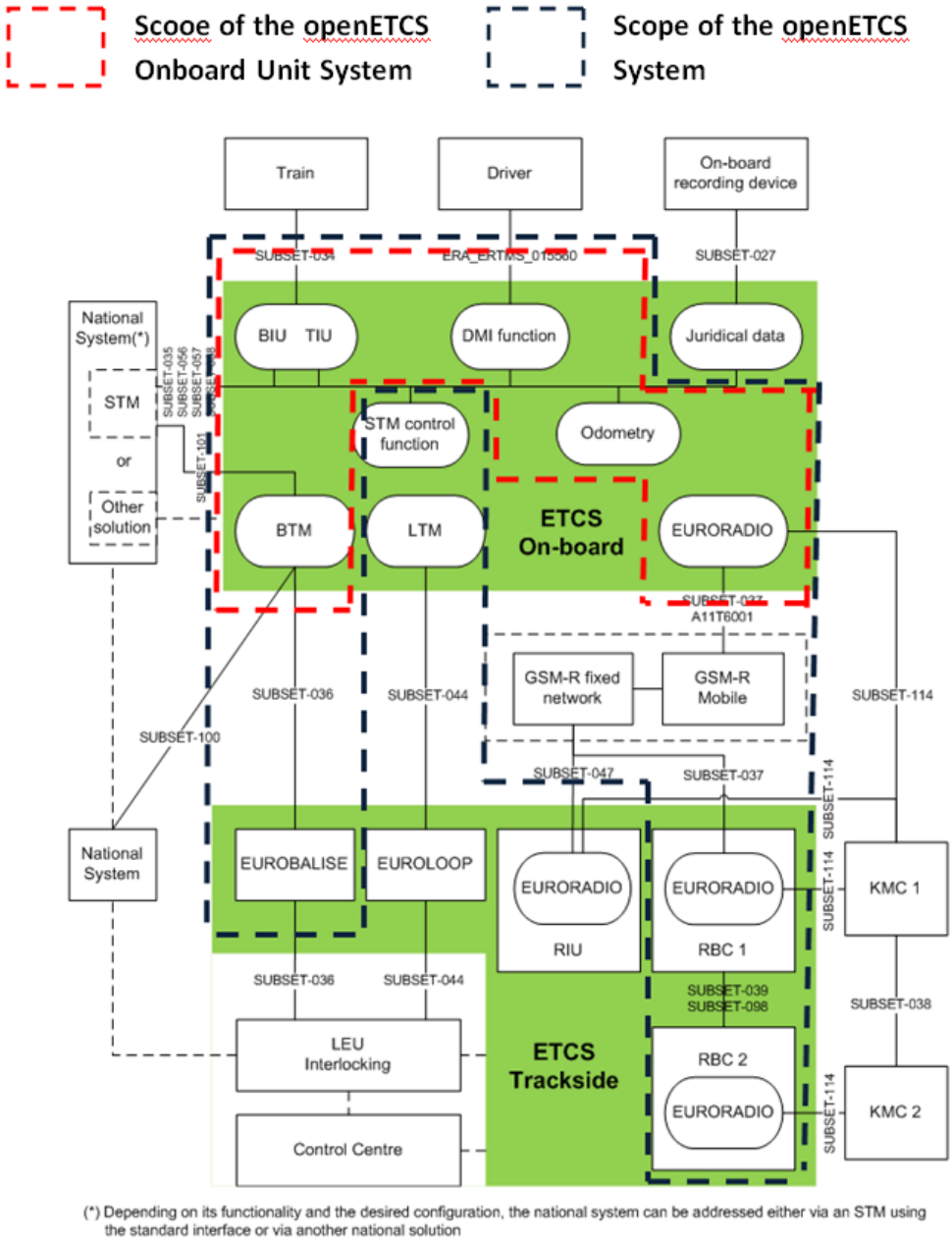


Figure 2. OpenETCS system of interest (dotted black line) to cover demonstration objective of ITEA2 project and according to ERA TSI Chapter 2

2.3 OpenETCS design process (WP3) and design artifacts

After recalling the development process and precising openETCS project scope we must now look at the design process to see which artifacts are produced to meet openETCS requirements. This process is under WP3 responsibility.

At system level (ETCS), we have to produce two main specifications: Elaborated System Requirements Specification and System Architecture Design Specification.

First idea would be to formalize SRS subset 026 on the scope defined previously and then continue system decomposition from this formalization but from experience of openETCS partners, this approach *is not sufficient in order to fully understand the main issues that stem from the approach by which the SRS was conceived*.

So, instead of building yet another copy-paste direct formalisation of the ETCS SRS, WP3 suggested another approach using a SRS functional analysis centred on Onboard unit (instead of EVC) and focused on the reference track (ETCS L2 Utrecht- Amsterdam).

This approach leads to the following architecture and design artifacts described in [**?**] :

- User story models that refine textual user stories about use of OBU in operational context: done with SysML language and Papyrus tool. It provides another entry for project stakeholder requirements and prepares validation.
- System Architecture model, defined with SysML language and Scade system Designer tool, with currently two decomposition levels:
  - System breakdown structure, centred on ETCS OBU and defining required interactions to other ETCS subsystems (external interfaces)
  - Functional breakdown from OBU, centred on ETCS Kernel and defining functional interactions between those internal functions (internal interfaces)
- OpenETCS application software executable model, a model defined with SCADE tool by integration of several functional block models that realize functional block interfaces identified and characterized in Architecture model
- Environment model, designed with SCADE tool and SCADE display for simulation HMI, that supports validation of OpenETCS application software executable model
- OBU SW Architecture and Design Document (ADD) providing a functional description of software architecture and design
- A RFC (Request For Comment) process that provides a cross-reference to the SRS, listing all requirements and design conflicts that have been identified during work.

Figure 3 illustrates the architecture breakdown performed in openETCS architecture model starting from ETCS system down to ETCS kernel software component.

Figure 4 summarises design process defined by WP3 and shows relations between different design documents and with reference input artifacts: SRS subset26, Utrech-Amsterdam reference track scenarios and openETCS API for platform interoperability. Black arrows mean "input for" while red arrows are traceability relations to reference artifacts.

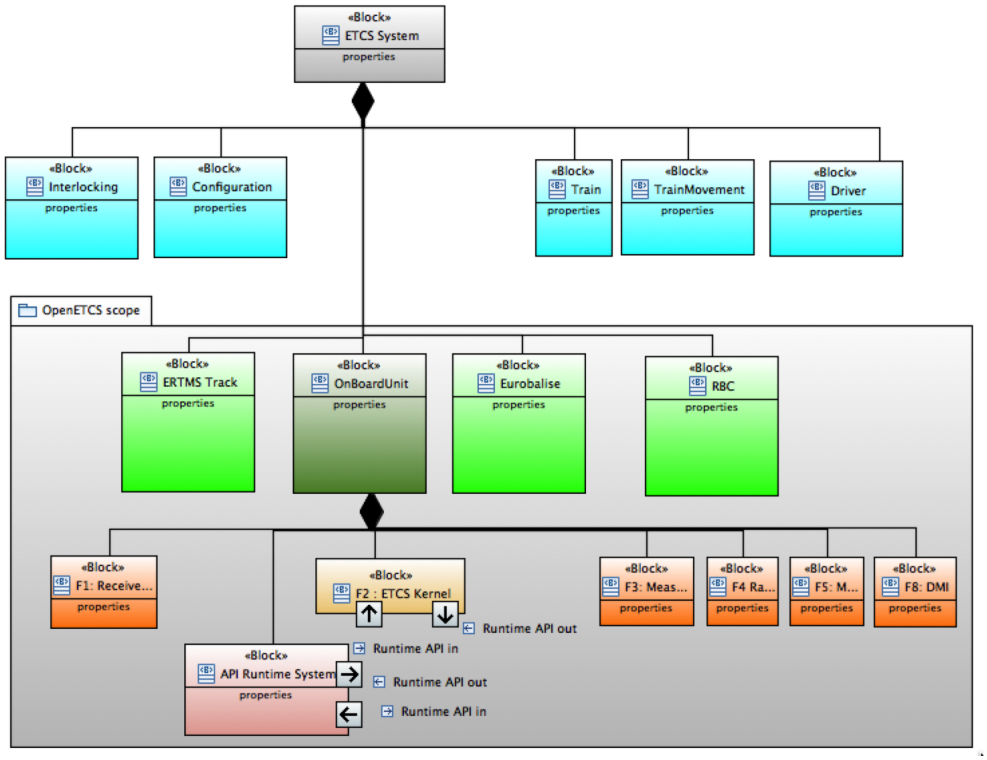


Figure 3. OpenETCS architecture with SysML: from system to software components

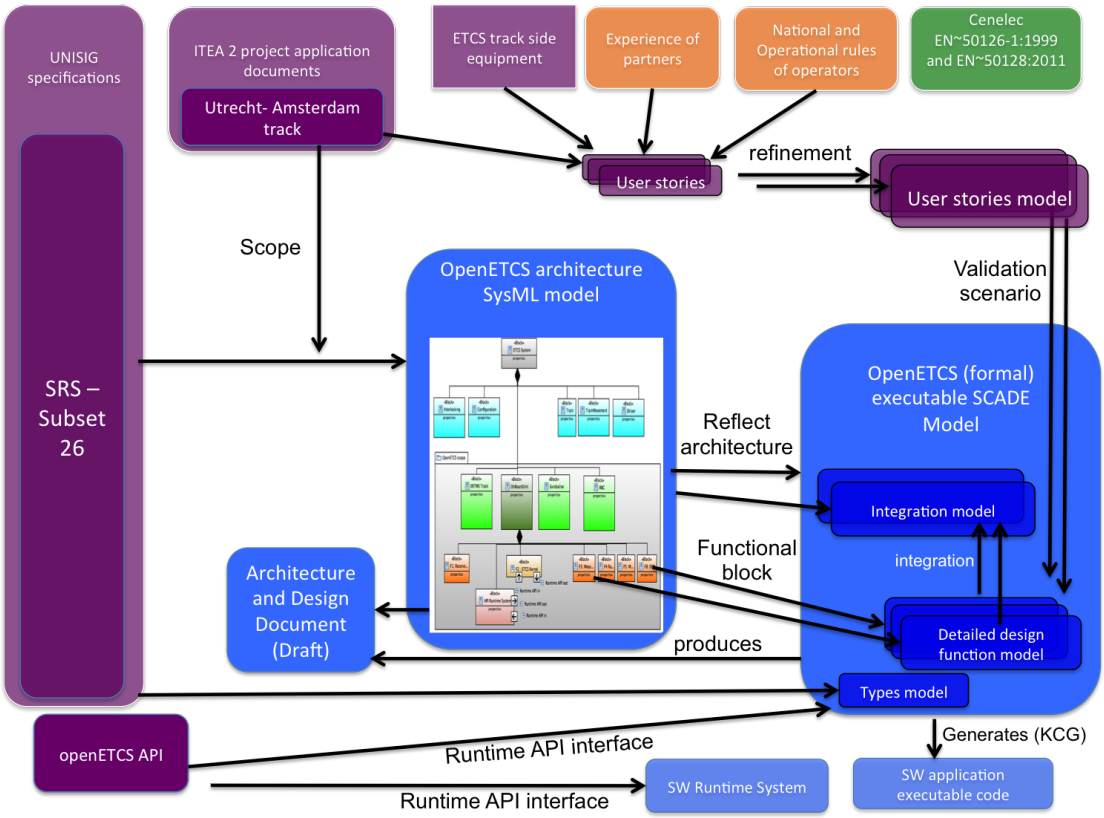


Figure 4. Relations between main WP3 Design Documents and reference input artifacts (SRS, Utrecht-Amsterdam reference track and OpenETCS API)

2.4 OpenETCS VV and safety process (WP4)

[5] recalls that the openETCS software development has to interact with a respective generic safety management process and take the overall safety requirements for a train control system into account.

During validation of product, OpenETCS project has to provide a safety case. The openETCS safety case shall present a transparent and easy to follow argumentation chain connecting the system definition with all basic safety assumptions and all evidence. To do this it has to show the relations between design and V& V artifact created during the iterative openETCS development process. Thereby, it has to be shown to which degree and under which assumptions the resulting documentation covers process, quality and safety requirements based on EN 50126, EN 50128 and EN 50129.

Figure 5 shows the interactions between design, verification and validation and the general quality and safety management.

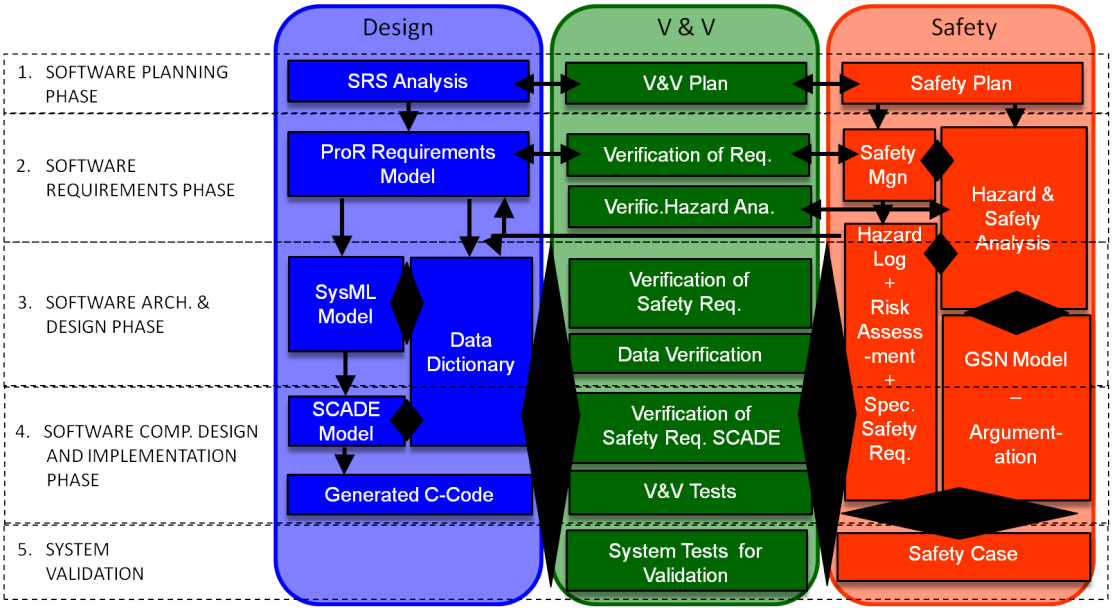


Figure 5. OpenETCS development relations between design, verification and validation and safety activities

3 OpenETCS traceability priorities and detailed expectations

3.1 Traceability main priority from WP3

From WP3 point of view, most important traceability chain concerns links between ETCS OBU formal model and Subset026 SRS requirements and all other requirements created during system analysis, either by decomposition, refinement or derivation. ETCS OBU model is detailed down to software detailed design level, and as it is a formal model, it becomes possible to generate code from that model. So, if it is possible to demonstrate that this model satisfies some Subset026 SRS requirements, then it will be possible to establish that software code generated from that model also satisfies those requirements.

Figure 6 highlights traceability chains with highest priority (arrows with largest size).

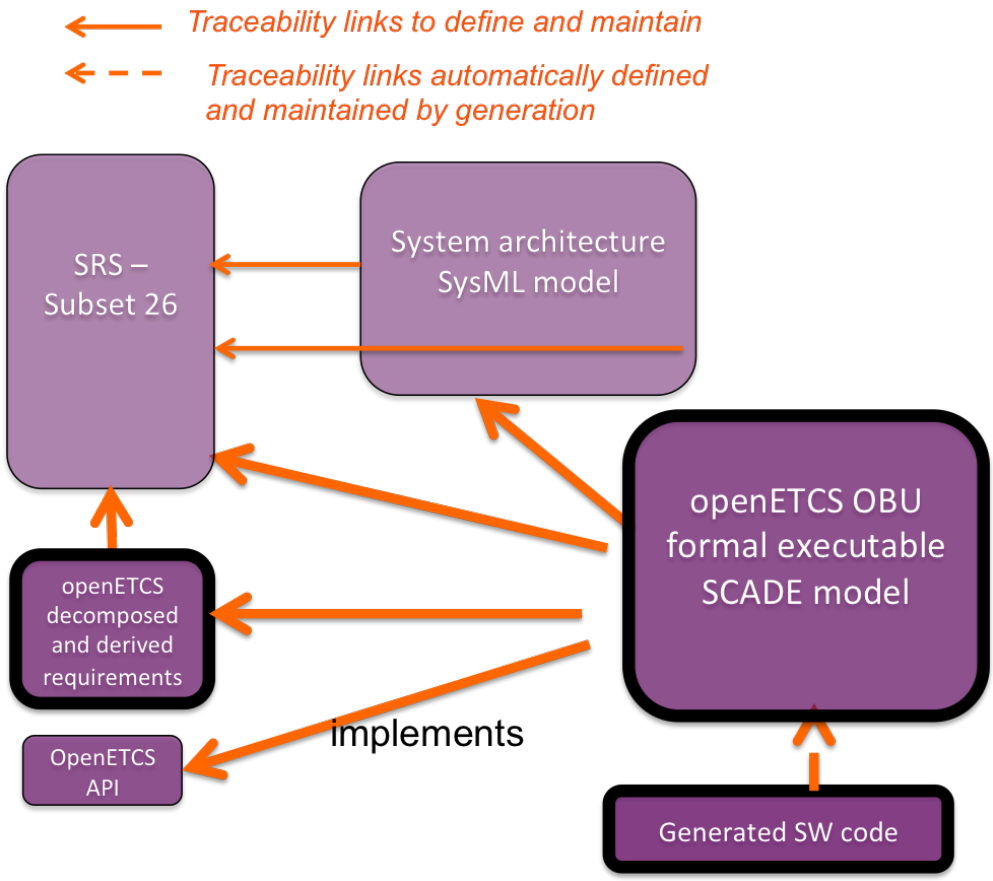


Figure 6. OpenETCS traceability chains with highest priority

Next paragraphs explain traceability during the 5 phases of the software development process with specific focus on links between openETCS OBU model and SRS Subset 26.

3.2 OpenETCS traceability main scenarios

Traceability main scenarios are grouped by phase according to the openETCS software development process introduced in previous chapter. We recall it below to ease reading of next paragraphs.

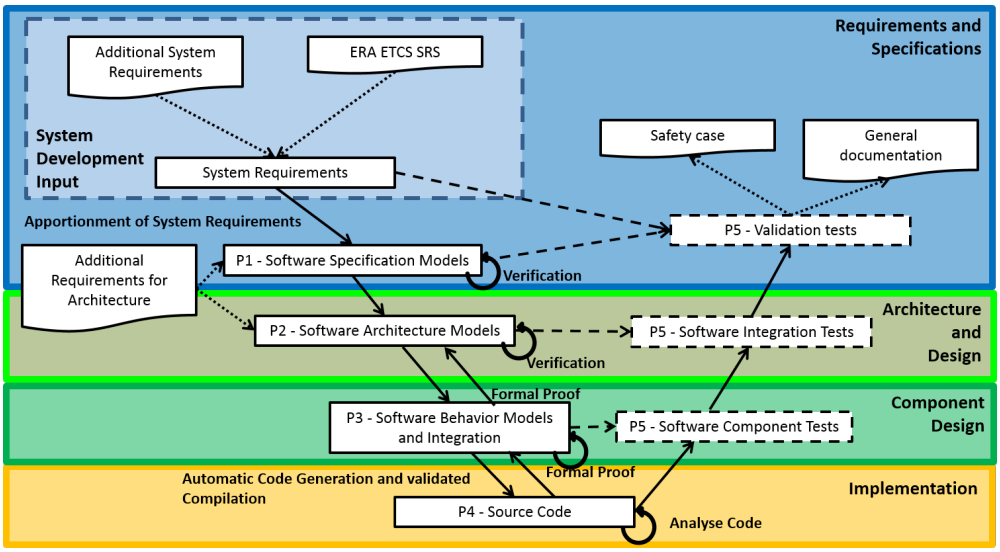


Figure 7. OpenETCS software development process

3.2.1 P1 Software specification Phase

P1 covers requirement development activities for software development. Figure 8 shows requirement derivation process from reference stakeholder requirements that make the OpenETCS baseline down to SW requirements.

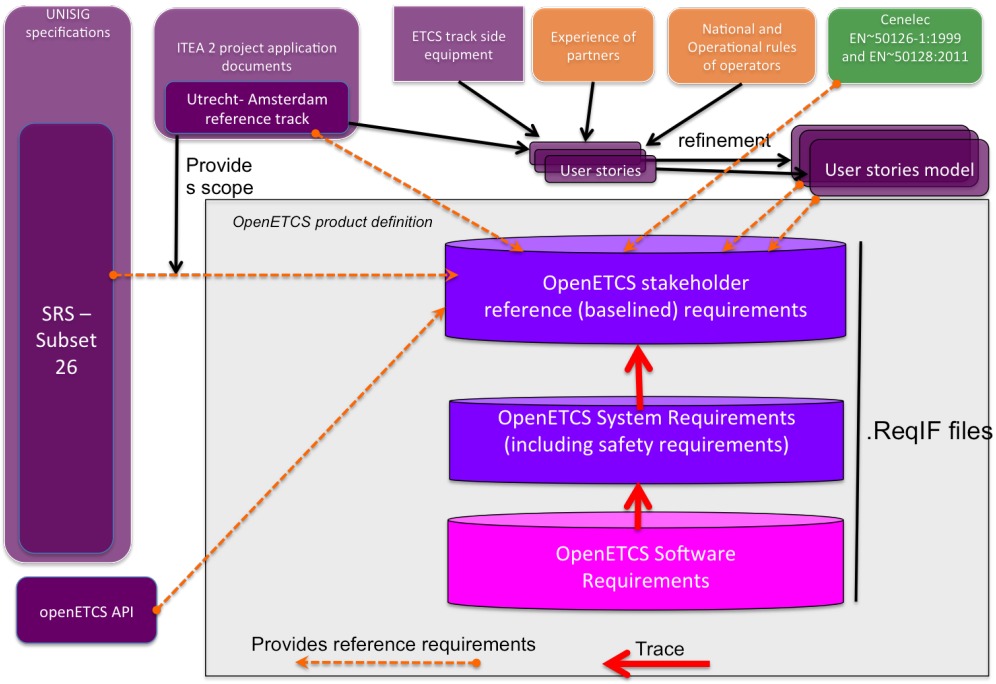


Figure 8. Requirement derivation process down to software requirements with traceability

Software Overall Test Specification will be derived with respect to software requirements, but a method is not defined at this point.

The main part of input system derivation process from SUBSET 26 to software requirements is done through SysML architecture model as illustrated in figure 9.

We do not detail this activity and associated traceability because it is not current priority. However, this derivation process remain important in order to ensure good relevance of software requirements with regards to SUBSET 26 and other system specifications.

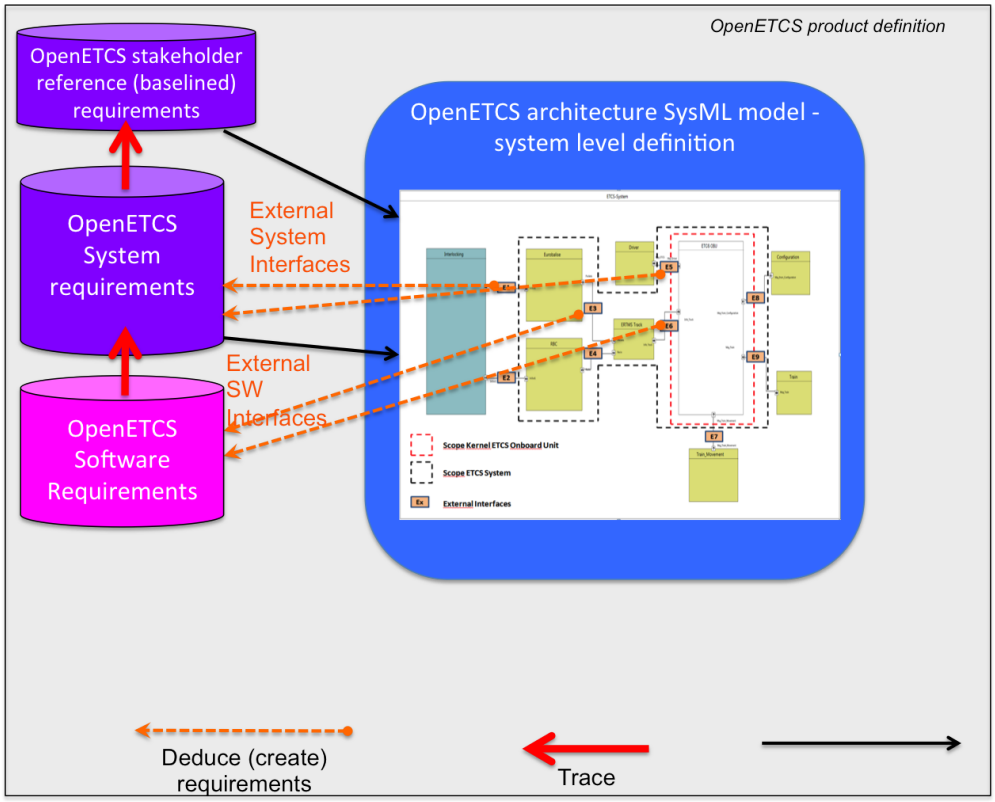


Figure 9. Requirement derivation process through SysML architecture model

3.2.2 P2 - Software Architecture Modeling Phase

The phase P2 covers activities from the EN 50128 development life cycle Software Arch. & Design Phase (7.3).

The SysML model initiated in P1 is completed to represent the Software Architecture specification. It is reflected into a SCADE model that will be used for integration (as the overall SCADE model is composed of several components described by SCADE models and the defined interfaces, the integration is performed and documented in the SCADE model).

Figure 10 summarizes those activities and associated artifacts.

3.2.3 P3 - Software Behavior Modeling and Integration Phase

The phase P3 *Software Behavior Modeling and Integration Phase* cover activities from the EN 50128 development life cycle Software Arch. & Design Phase (7.3), Software Component Design Phase (7.4) and Software Component Implementation Phase (7.5). The SCADE model is refined and enhanced to completely represent Design and Interface Specifications as well as all

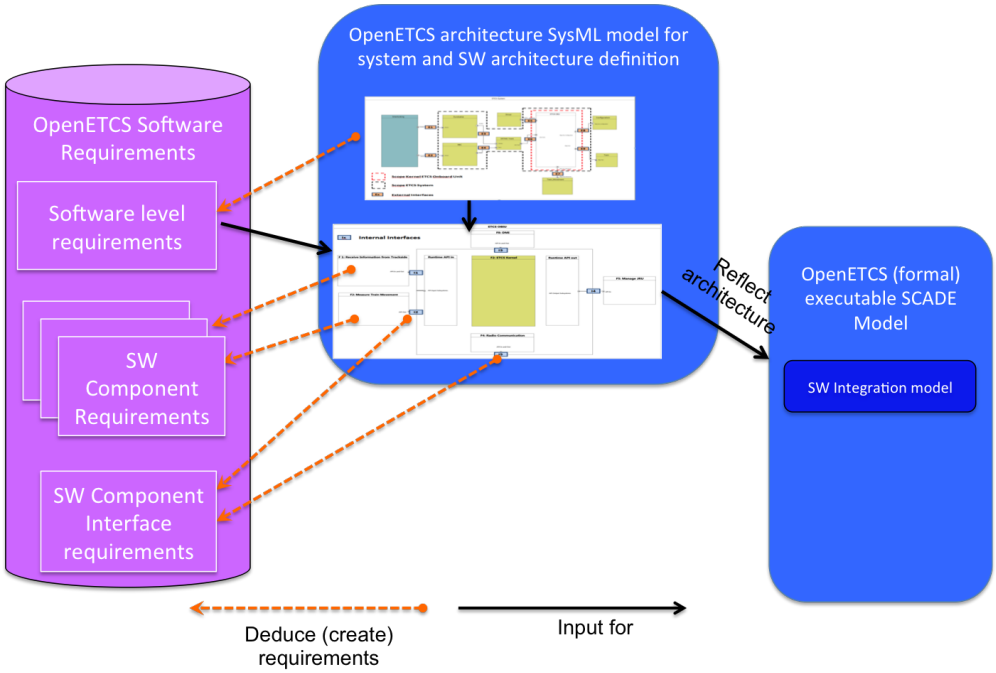


Figure 10. Software architecture definition through SysML model

Component Design Specifications. Source Code in C is generated automatically via the certified SCADE Code Generator and combined with required wrapper software. For these wrapper parts specific development methods have to be defined as these parts are specified.

Figure toDo.●

3.2.4 P5 - Formal Validation Phase

The phase P5 has to cover activities of the Software Component Testing Phase (7.5), Integration Phase (7.6) and mainly Software Validation Phase (7.7) by using simulation, testing and model checking methods. To do so the SCADE environment as well as additional tools are use depending on the specific model property to be tested or proven.

The Inputs for these activities are the SysML and SCADE models and the C Code.

4 OpenETCS tool chain requirements about traceability

Requirement traceability cannot be fully supported by tool chain if there is no ability to manage requirements (create, edit, visualize, classify...) as expressed in previous chapter through traceability process and main scenario. Next paragraphs define tool chain capabilities required to support requirement management and requirement traceability in the context of OpenETCS expectations presented previously.

4.1 Tool chain capabilities to support requirement management

Requirement management is an activity that consists in supporting different tasks concerning requirements during the project, whatever the engineering level. In OpenETCS project we need at least the following commands available in the tool chain (and potentially restricted according to access rights if defined):

- Visualization of one or several requirements, their links if any and their attributes.
  - **OpenETCS example**: visualize one Subset026 SRS or API requirement, its statement and its hierarchy (father requirement if any and children requirements if any).
- Query (filter, ordering) on a set of requirements with filter based on requirement attribute values or on links between requirements
  - **OpenETCS example**: list only subset026 SRS chapter 7 requirements that are not yet traced (no traceability link).
- Creation and storage of requirement and of different attributes based on a given requirement template/type, in a given requirement hierarchy and with unique identifier allocation based on a flexible (customizable) strategy
  - **OpenETCS example**: creation of a new openETCS requirement decomposed from a susbet026 SRS requirement. Newly created requirement shall have unique identifier automatically allocated to be consistent with its hierarchy (father requirement id) and with an attribute "maturity" set to "to be confirmed" and creation date set to the current date.
- classification of one or several requirements into modules/groups that can be defined by end users.
  - **OpenETCS example**: allocation of a newly created requirement to the Onboard Unit scope.
- Edition of one or several requirements and their attributes
  - **OpenETCS example**: confirmation of a newly created requirement with attribute "maturity" set to "confirmed".
- Addition of a version on one requirement or on a set of requirements

  - **OpenETCS example**: selection of a set of reviewed openETCS requirements and addition of a version for all those requirements.

- Ability to compare two versions of a set of requirements and list all requirements that have been added or modified.

  - **OpenETCS example**: comparison of two versions of Subset026 SRS requirements and visualization of all requirement links to check.

- Import of requirements coming from external sources (ReqIF, Word, Excel...)

  - **openETCS example**: import of Subset026 Word document or ReqIF format. Import of openETCS API definition.

- Export of a set of requirements to another format: at least ReqIF standard interchange format but also office format (.csv, excel or word...)

  - **OpenETCS example**: export of all openETCS created requirements and links into .csv file

In addition, requirement management tooling shall enable share and access of requirements within a team, through a shared repository (shared directory on a network drive or CVS repository like SVN, Git, ClearCase or any other one): either directly (all team members access same shared repository) or with local work and synchronizations between team members through a shared repository.

4.2 Tool chain capabilities to support requirement traceability

Requirement traceability activity consists in ensuring that all product engineering artifacts (including verification means) can be traced to an originating stakeholder requirement either directly (direct link) or through other system requirements derived from stakeholder requirements. It means creating links but also manage their status (created, confirmed...) and potentially their deletion.

In order to support this activity in openETCS project we need at least the following commands available in the tool chain:

- Creation of a link between a requirement and an engineering artifact, based on a given link template/type (refine, derive, implement, verify...).

  - **OpenETCS example**: create a "Satisfy" link between one Subset026 SRS requirement and one OnBoard Unit function, with "status" link attribute set to "defined" and "rationale" attribute set with appropriate justification.

- edition of link status.

  - **OpenETCS example**: after review, confirm some traceability links by setting "status" attribute to "validated" value.
  - **other OpenETCS example**: after change in some Subset026 SRS requirements, for all traceability links of modified requirements, set status to "to check" value.

- deletion of requirement traceability link.

  - **OpenETCS example**: after review, decide that some traceability are not accurate and delete them.

- export of requirement traceability
  - **OpenETCS example**: after progress meeting, export current requirement traceability to .csv file so that it can be analysed by project manager and/or quality team

5 Tool chain logical architecture and current solution to support traceability

5.1 Components and their interactions

[1] presents in table 1 a summary of the 5 phases of the openETCS software development process and lists the different tools used to support the 5 phases .

Table 1. Phases openETCS software development process

Phase	Name	Description	Main Tool component
P1	Software Requirement Phase	Requirement input documents converted to a ReqIF format and informally analyses. The analysis specifies relationships between requirements and revises parts of the requirements to obtain a detail and atomic abstraction level usable for moralization. To support thus the requirements are categorized and grouped.	ProR
P2	Software Architecture Modeling Phase	Building a SysML based on the informal analysis using the categorization of requirements. The architecture model focuses on functional blocks and data flows.	SysML Papyrus
P3	Software Behavior Modeling and Integration Phase	Building the SCADE model for the separated basic functional blocks. The SCADE model describes the detailed behavior for the function using the data flows.	SCADE Suite
P4	Code Generation Phase	Based on the SCADE Behavior Model C code will be automatically generated. This C code is then compiled to executable code which runs on the EVC.	SCADE Suite + C Code compiler
P5	Formal Validation Phase	Using test models and model checking techniques, to validate the correct model behavior.	Various tools

Figure 11 shows current tool chain architecture with two components(in red) identified to support respectively requirement management and requirement traceability.

Figure 12 shows interactions between tool chain logical components concerning requirement flow down and traceability.

5.2 Component design: existing technologies/tooling

5.2.1 Requirement management and traceability

Concerning requirement management and traceability there exist a lot of solutions. If we focus on open source solutions and especially those able to integrate into Eclipse platform, we find:

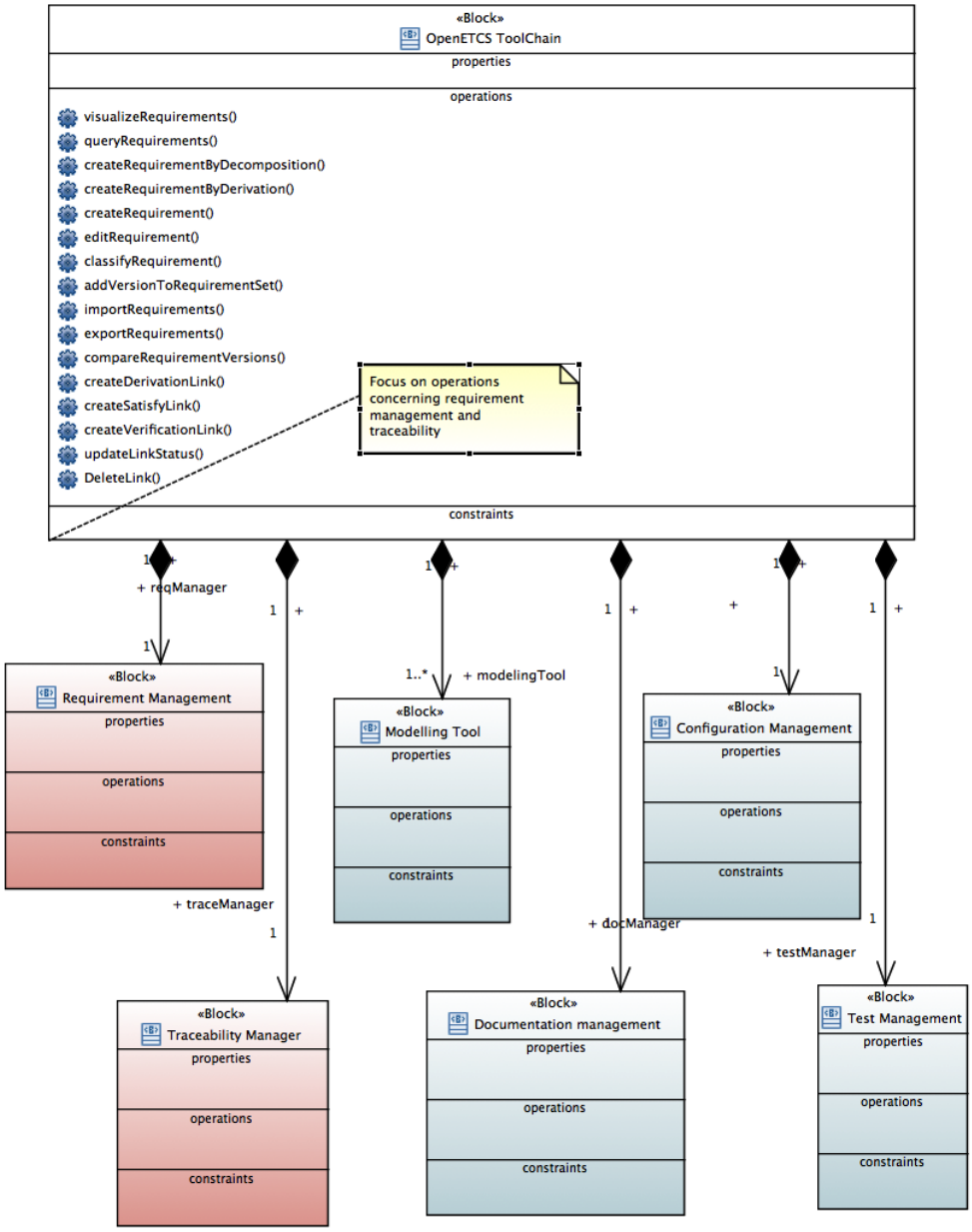


Figure 11. Tool chain logical architecture with focus on requirement management and traceability

- Eclipse Requirement Modeling Framework (RMF). As mentioned on Eclipse web site, RMF is a framework to manage textual requirements through ReqIF standard requirement exchange format. There is a powerful GUI called "ProR" to configure, visualize and edit ReqIF requirements in RMF. RMF/ProR is the natural choice for OpenETCS to support requirement management.

  Alternatively to deal with the closed source path the requirement management may be done by Reqtify Gateway of SCADE (also called "RM Gateway").

  ProR can also be used to support traceability between requirements (native functionality) and between requirements and Papyrus models through a "proxy" connector.

- PolarSys ReqCycle. It is a solution that focuses on requirement traceability. It can allow managing requirements (creation, edition, queries) with classification (scopes) and custom requirement data models but with very limited requirement editor. Strength is its ability to capture and aggregate traceability links coming from different sources (EMF models, C code,

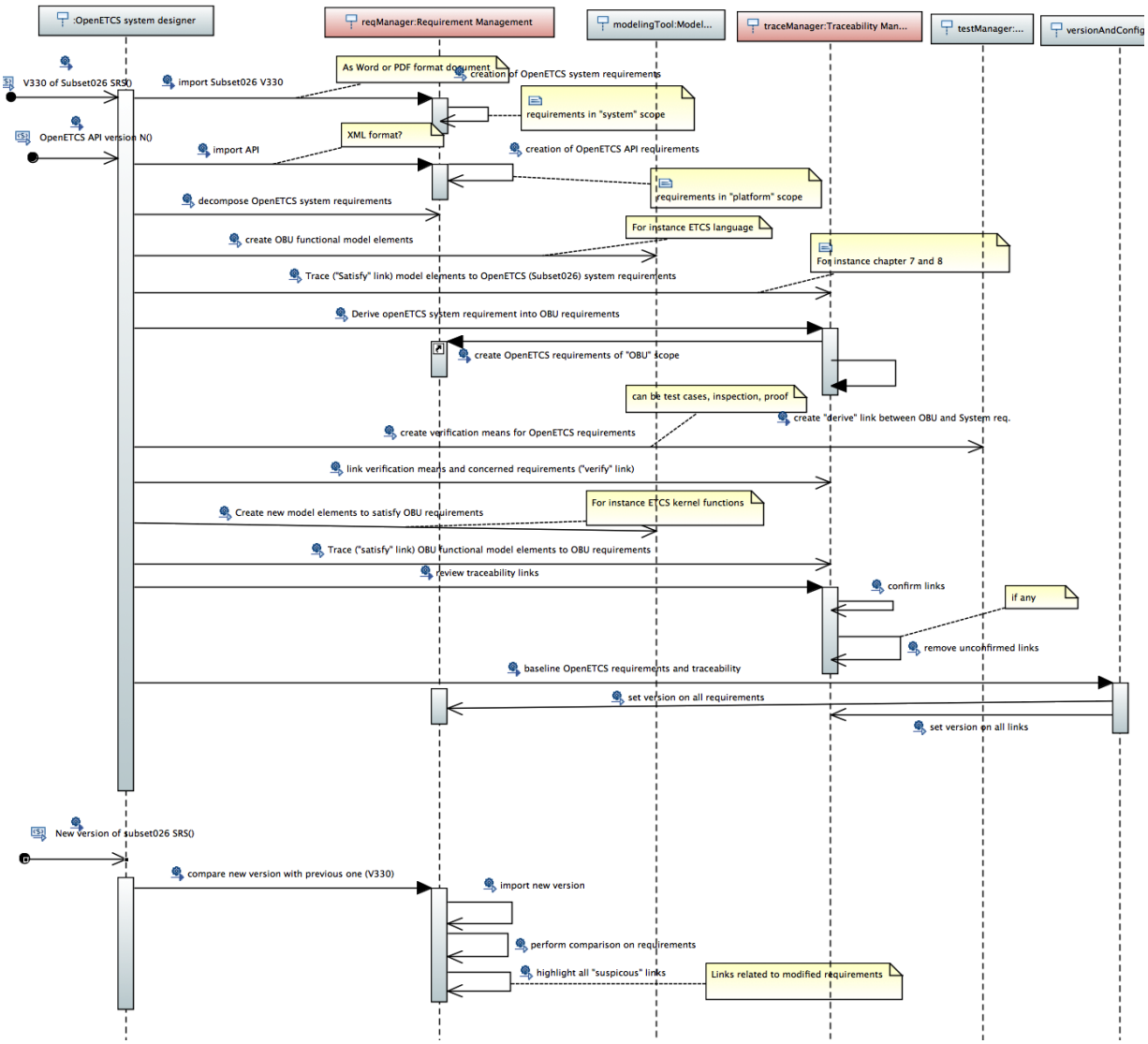


Figure 12. interaction between tool chain components to support requirement flow down and traceability

Java code, SysML models...) and to define own traceability links based on custom link types with potential attributes.

If we extend search to proprietary solutions, we find IBM DOORS and IBM DOORS Next requirement management products and Dassault Systems ReqTify product as leading solutions to support respectively requirement management and requirement traceability.

5.2.2 Modelling tool for system model

The system model defines block and interfaces between those blocks that realize the specification. This is done with Eclipse Papyrus.

5.2.3 Modelling tool for OBU functional model

The system model defines block and interfaces between those blocks that realize the specification. This is done with Scade Studio (V16.2)

5.2.4 OBU Functional model

OBU functional model is done with SCADE suite tool.

6 First physical traceability solution: ProR-RMGateway

This first solution consists in handling traceability "by hand" (without full integration in the tool chain), with different solutions locally integrated to the different modelling environments used in the project (SCADE and Papyrus) and aggregate results in a shared global requirement database.

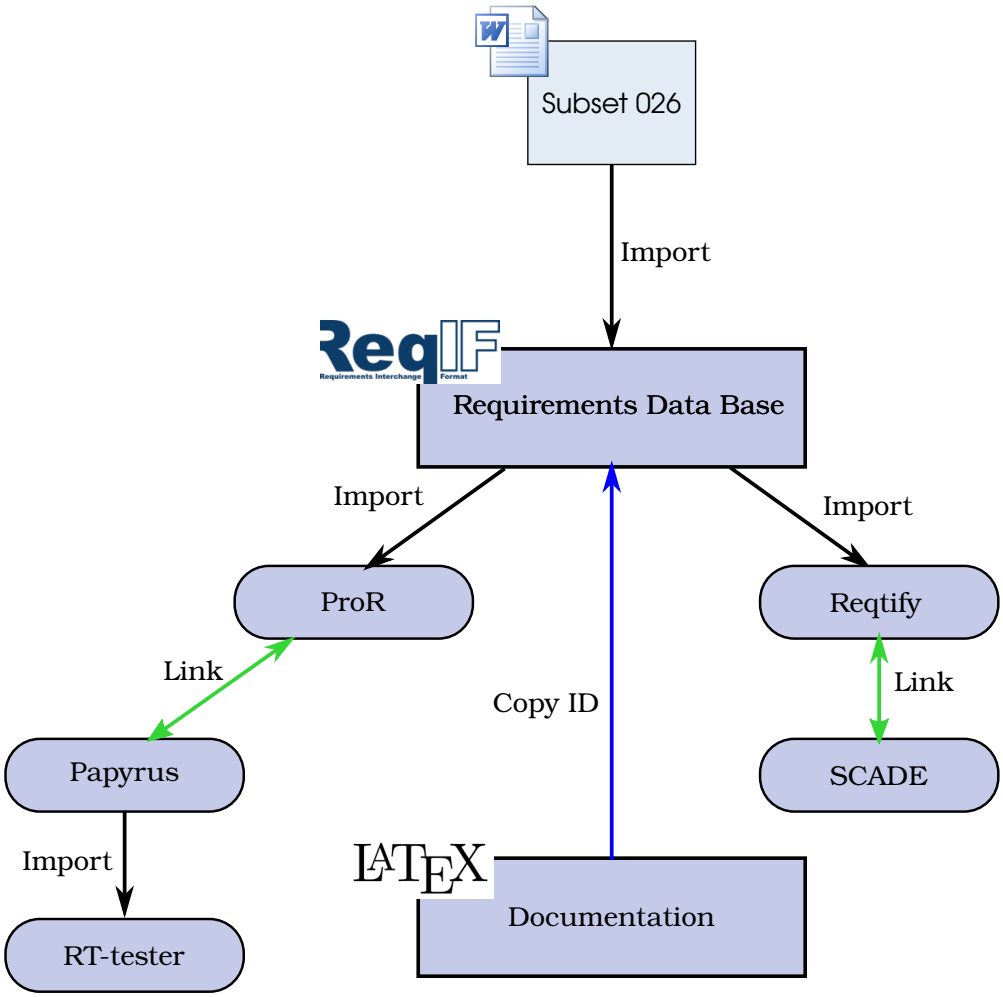


Figure 13. Traceability architecture first solution

With that approach, there is one master reference requirements data base. This data base is itself directly imported from the subset-026 word document. The data base provides the list of requirements as well as an unique identifiers for each requirement. These identifiers are fixed and cannot be modified by any other tools.

Note: it is possible for requirement management tools to complete imported requirements but they should guarantee the following rules:

1. The requirement's structure (hierarchy, scope) is not modified.
2. No requirement should be deleted.

3. Newly added requirements should be either a refinement, derivation or a decomposition of an existing one.
4. The identifier pattern should be respected.
5. Identifiers should remain unique.

Concerning database, best solution is to use ReqIF standard exchange format, as it can be fulfilled by most requirement management tools.

Concerning requirement management and traceability "local" solutions we get:

- For Scade modelling: most direct solution is Scade RM-Gateway (based on ReqTify solution) as it is integrated with SCADE tool. Note: alternative solutions are "ReqCycle" (see other solutions later in this document).
- For Papyrus SysML model: ProR can be used with a given proxy added to Papyrus to support creation of links between ReqIF requirements and SysML elements. Note: another direct solution is to use ReqCycle solution that is well suited to create and capture links with Papyrus models. It will be investigated in another candidate architecture.

  Figure 13 illustrates this solution with associated tools/technologies.

6.1 Subset-026 import

The subset-026 import is realized by a script transforming the Word document into a req-IF format file.

6.1.1 Script description

The script generates a hierarchical tree of all traceworthy artifacts in each chapter of subset-026. Each artifact shall be uniquely addressable via a tracestring.

The script needs a name

6.1.2 Unique ID definition

Take the following example:

3.5.3.7 If the establishment of a communication session is initiated by the on-board, it shall be performed according to the following steps:

a) The on-board shall request the set-up of a safe radio connection with the trackside. If this request is part of an on-going Start of Mission procedure, it shall be repeated until successful or a defined number of times (see Appendix A3.1).

If this request is not part of an on-going Start of Mission procedure, it shall be repeated until at least one of the following conditions is met:

- Safe radio connection is set up
- End of Mission is performed
- Order to terminate communication session is received from trackside

Figure 14. Traceability architecture between artifacts

Guideline

The scope of a single requirement ID is a paragraph of text (there are six such paragraphs in the above example). requirement IDs are hierarchical. The hierarchy is a direct mapping of the hierarchy in the original subset-026 text. Levels are separated by a dot. There is a requirement at each level (i.e. you may truncate the requirement ID to any level and it stays valid).

How to

Suppose we want to trace the fifth paragraph in the above example i.e

- `End of mission is performed`

1. Let *traceString* be the variable to store the result.
2. Find the current running number of the base list. That is the list which includes the chapter number. In this example this number equals `3.5.3.7.` Set *traceString* to this number.
3. Count the number of paragraphs in this list item starting with 1 and append this number in square brackets to the *traceString* if it is greater than 1.

   Note: For the first iteration in the example there is only one such paragraph (`If the establishment...`). Hence, we do not append anything. In the second iteration there are two such paragraphs (`The on-board shall...` and `If this request is not ...`). Hence, the second one will receive an `[2]` appendix.
4. Until you arrived at your target paragraph: Append any running number of sub-lists and remove leading or trailing characters (such as braces). If the current sub-list is bulleted then the level string always becomes `[*][n]` (with n being the running number of that bullet starting at 1). Prefix this new level with a dot (`.`) and append it to the *traceString*.

   Note: `a)` is the identifier of one such sub-list item. The trailing brace will be removed. The bullet points form another (less significant) sub-list.
5. Do step 3.
6. Do step 4 or break.
7. *traceString* is now the fully qualified requirementID.

This will result in the following requirement ID: `3.5.3.7.a[2].[*][2]`

6.2 System Model

The link with the requirement may be included via requirements diagram with a direct link of requirement in ProR The explanation to performs the link may be found here ProR-Papyrus proxy. The links may be viewed through Papyrus an ProR and the requirement may be directly apply from ProR to a Papyrus elements.

6.3 Interface Definition

It should define the interfaces between the architecture artifacts. It is used and set up to facilitate team working together on a big architecture. Its definition comes from the requirements but can also be refined by the modelling team without changing the existing implemented requirements.

6.4 TODO Verification and Validation

7 Second physical traceability solution: ProR-ReqCycle

This second solution consists in using only one centralized Requirement database (as in solution 1) managed by one eclipse-based requirement management solution (ProR) and only one eclipse-based technology to support requirement traceability for all models (Papyrus and SCADE): PolarSys ReqCycle.

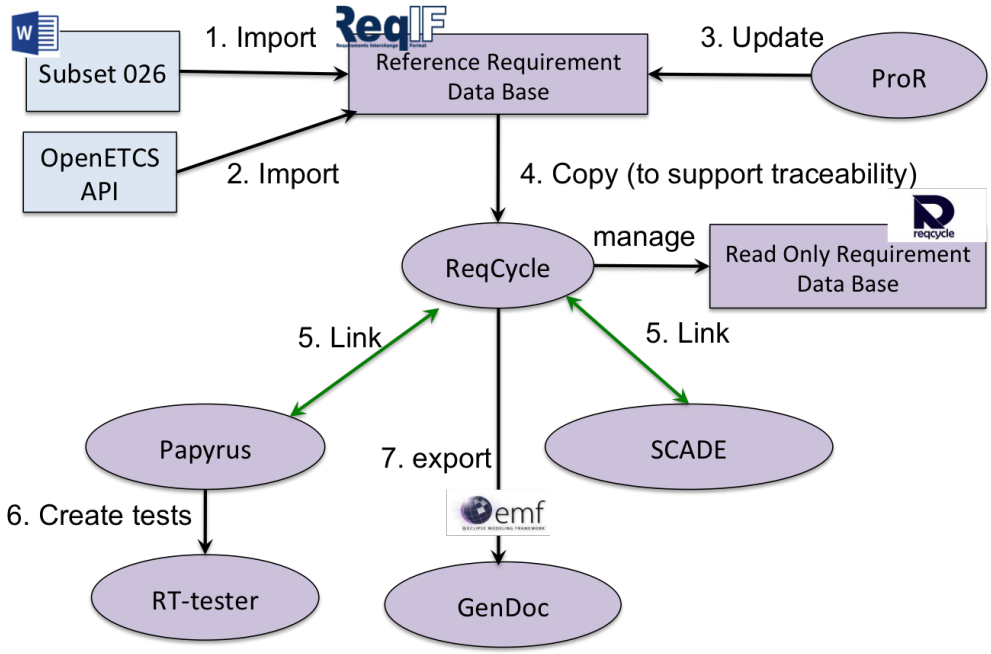


Figure 15. Traceability architecture second solution

With that approach, there is still one master reference requirements data base entirely managed by ProR. Requirement database is initialized by import from the subset-026 word document, as in solution 1 (see 6.1) and by import from OpenETCS API. All new OpenETCS requirements are added in this requirement database through ProR tool.

Traceability is managed by ReqCycle tool. In order to ease visual selection of requirements in ReqCycle, there is a copy (could be a link) of reference requirements hierarchy into a ReqCycle database. Then ReqCycle manages links with Papyrus and links with SCADE. Finally, traceability can be exported by ReqCycle and processed by Gendoc tool to deliver documentation.

Interactions through the tool chain are detailed in figure 16 below with focus put on traceability between requirements and functional model (traceability with SysML model, creation of tests and export of traceability are not shown here).

Note: this solution requires synchronization from ProR to ReqCycle each time requirements change in the reference.

Most complex commands are detailed in next paragraphs.

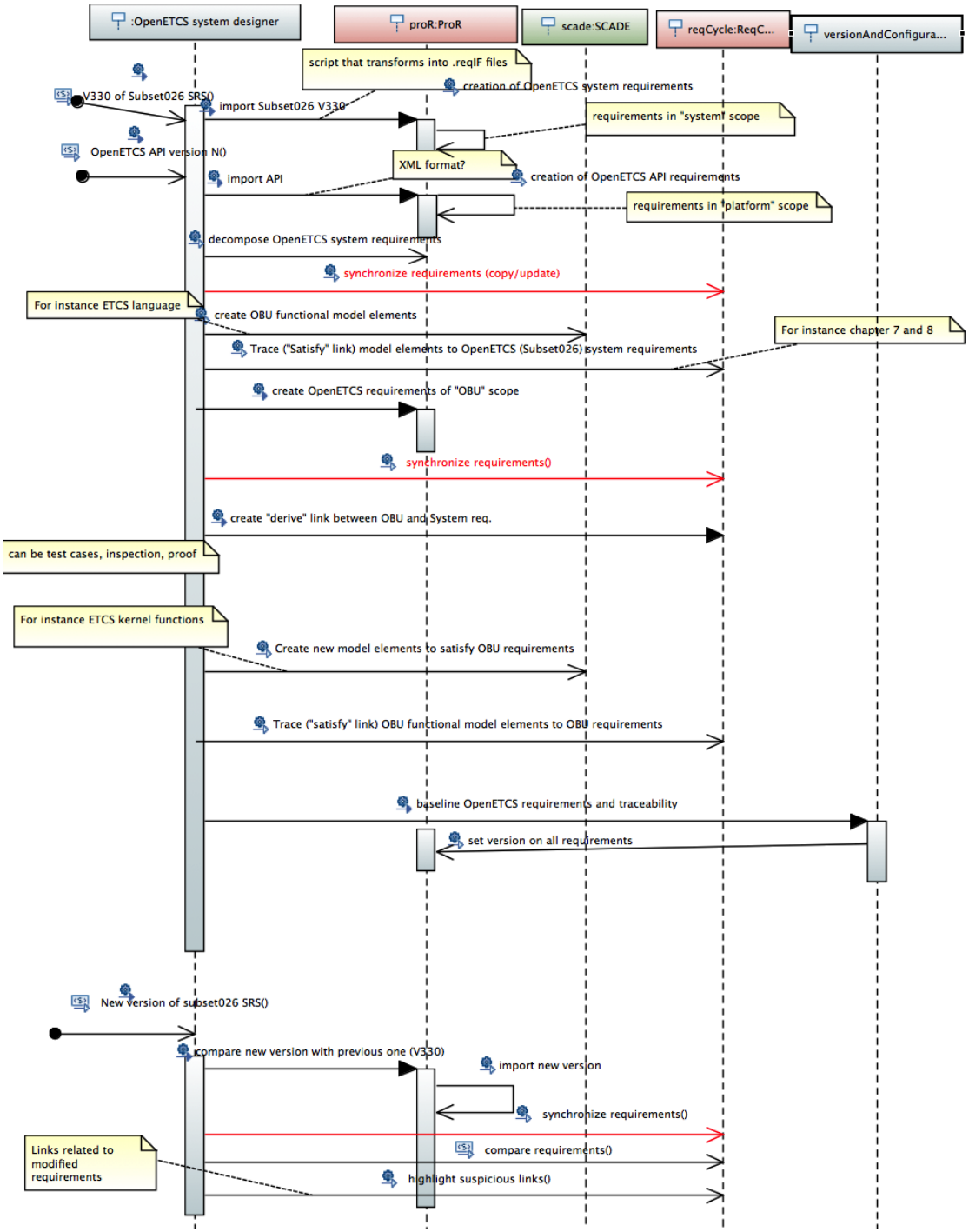


Figure 16. second solution - interactions between tool chain components

7.1 Subset-026 import

The subset-026 import can be realized by two means:

- Either by direct import from the subset-026 word document (ReqCycle Document import connector)
- Or by a two steps process with transformation from word document to .ReqIF files (1.B1 path in the figure) and then import of those .ReqIF files into ReqCycle requirement database (1.B2 path).
  - 1.A1 transformation can be realized with same script than the one described in first traceability solution - see 6.1
  - 1.A2 transformation (import) can be done by ReqCycle ReqIF import connector.

7.2 Creation of additional OpenETCS requirements

ProR can create new requirements by decomposition or by derivation of existing requirements. In case of derivation, engineering level (scope) changes (for instance from System to OBU or to OBU Kernel).

Regularly, when there have been new requirements created, there must by a synchronization from ReqIF reference requirement database to ReqCycle internal database so that ReqCycle can manage traceability to those requirements. This synchronization is done by a ReqCycle "import or update" command from .reqIF files: first time it is an import (creation of requirements) and next times this is an update of existing requirements with ability to highlights associated traceability links (that have to be checked).

Note: in the future, we could have a link (reference) between ProR requirement database (ReqIF format) and ReqCycle database, rather than a copy.

We could also have traceability driven by ReqCycle but selection of requirement done in ProR tool.

7.3 Creation of links between SysML model elements and requirements

ReqCycle provides a view that allows creating a link between two selected objects (traceability Creator). See figure 17.

Concerning selection of requirements, ReqCycle provides a view (Requirement View) from which it is possible to select one source object. Once selection is done, it can be propagated in traceability creator view (source area).

Concerning selection of SysML model elements, Papyrus provides a view with model elements (Model explorer) from which it is possible to select a destination object. Once selection is done, it can be propagated to the traceability creator view (destination area).

When both source and destination objects are selected, ReqCycle provides a command (button) to create links between selected objects if a link type definition is compliant with such source and destination (Refinement in the illustration on figure 17 below).

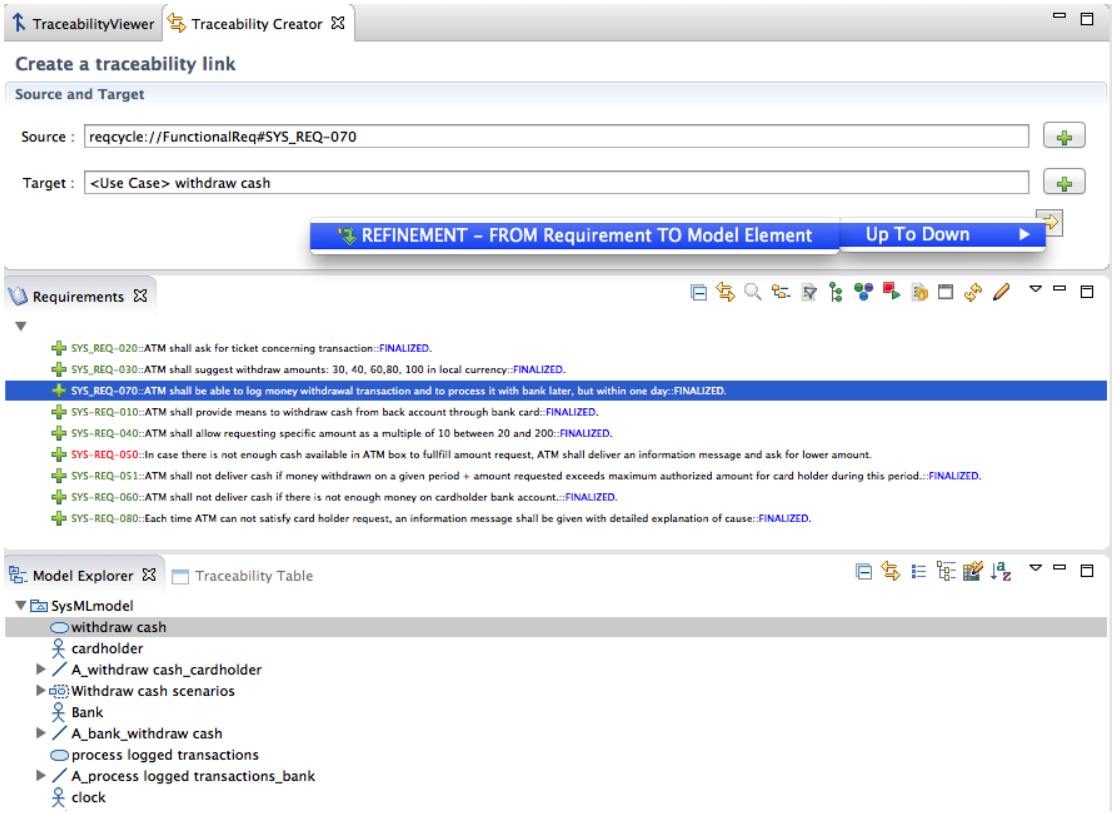


Figure 17. ReqCycle - traceability creator

7.4 Creation of links between SCADE model and requirements

There are two approaches that can be used:

1. use same approach than for creation of trace links between SysML model elements and OpenETCS requirements (see 7.3).

   Note: there is some investigation to check that it is (easily) feasible to propagate selection of a SCADE model element in ReqCycle traceability creator view (source or destination area).

2. use SCADE comment area available for any model element, fill it with traceability link data and let ReqCycle capture those traceability links by analysing SCADE model file (dedicated SCADE traceability analysis engine to implement).

   In order to avoid mistakes on editing link annotation in comment area in SCADE, idea is to generate requirement traceability link annotation from ReqCycle command ("Generate requirement traceability annotation for external element") and copy it into the clipboard.

   Then, in SCADE, system designer has just to "paste" string (from clipboard) in the concerned model element comment area and let ReqCycle analyse traceability: ReqCycle will parse SCADE model file, will capture all comments with associated model elements and will build the traceability links from requirement ID, trace link type and SCADE model element.

   Figure 18 illustrates such command.

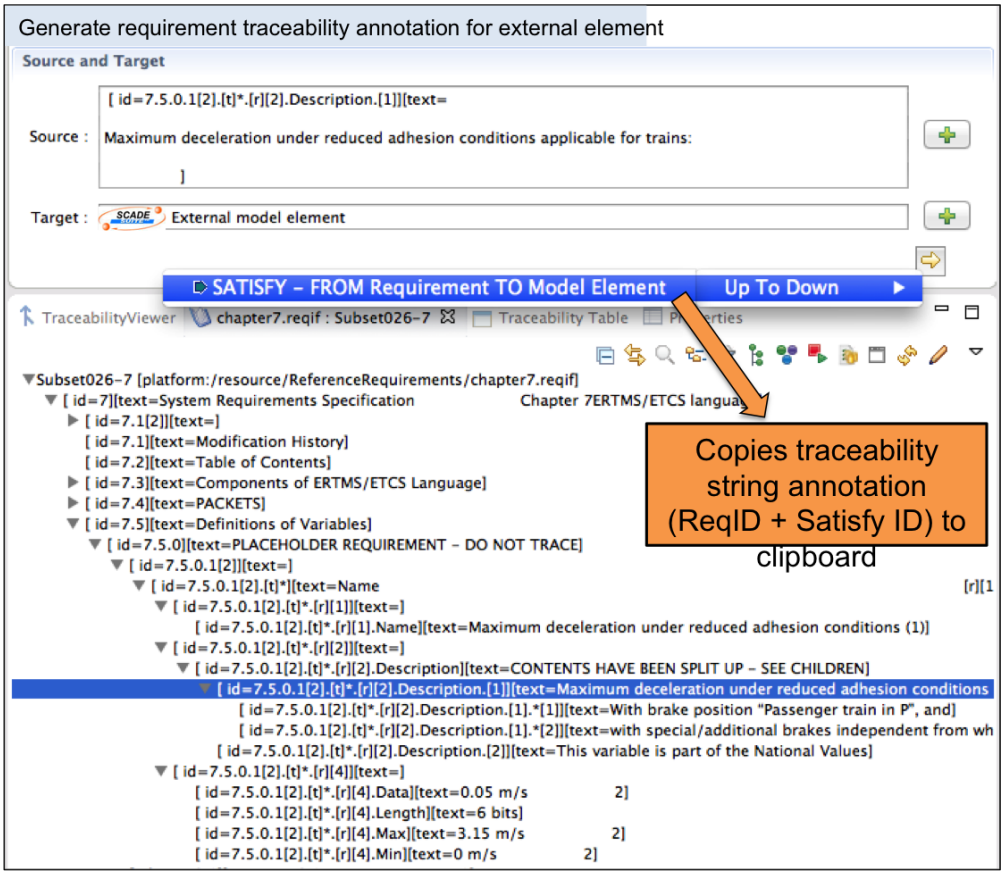


Figure 18. ReqCycle - generate traceability annotation for external model element (SCADE for instance)

7.5 Aggregation of traceability links and export

It is done by ReqCycle that has all information (copy of the requirement database and associated traceability between requirements + all traceability links with models). It is exported into EMF format that can be processed by Gendoc to render expected documentation including traceability.

8 Third physical traceability solution: ReqCycle

This third solution consists in using only one centralized Requirement database (as in previous solutions) managed by one eclipse-based solution (ReqCycle) also used to support requirement traceability for all models (Papyrus and SCADE).

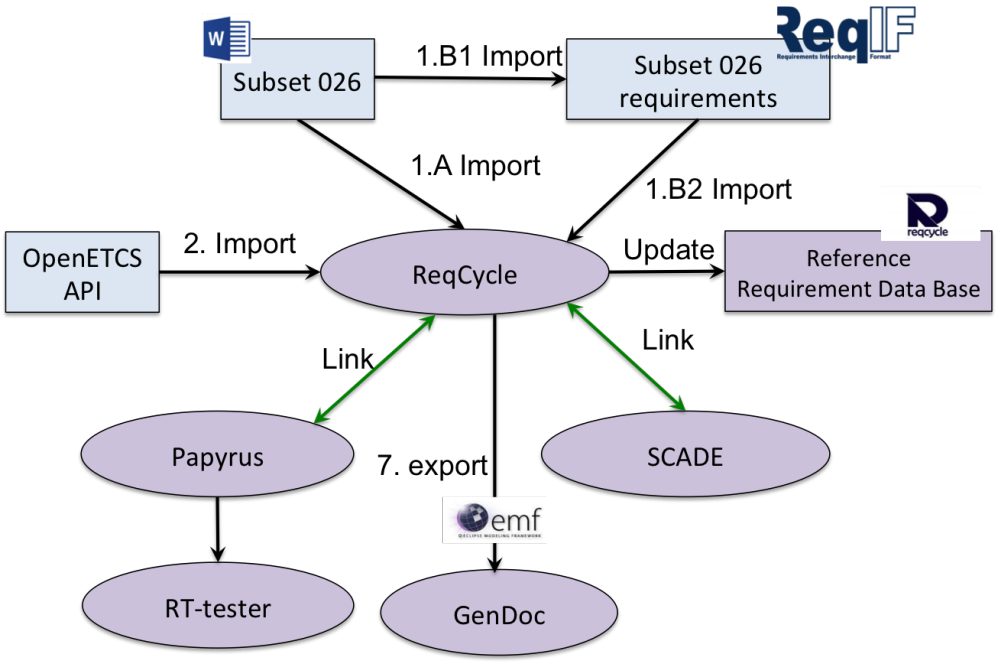


Figure 19. Traceability architecture third solution with ReqCycle only

With that approach, there is still one master reference requirements data base entirely managed by ReqCycle. Requirement database is initialized by import from the subset-026 word document through two possible means (see next section) and by import from OpenETCS API. All new OpenETCS requirements are added in this requirement database through ReqCycle tool.

Traceability is managed by ReqCycle tool. In order to ease visual selection of requirements in ReqCycle, there is a copy (could be a link) of reference requirements hierarchy into a ReqCycle database. Then ReqCycle manages links with Papyrus and links with SCADE. Finally, traceability can be exported by ReqCycle and processed by Gendoc tool to deliver documentation.

Most complex commands are detailed in next paragraphs.

8.1 Subset-026 import

The subset-026 import can be realized by two means:

- either by direct import from the subset-026 word document (ReqCycle Document import connector)
- else by a two steps process with transformation from word document to .ReqIF files (1.B1 path in the figure) and then import of those .ReqIF files into ReqCycle requirement database (1.B2 path).

- 1.A1 transformation can be realized with same script than the one described in first traceability solution - see 6.1
- 1.A2 transformation (import) can be done by ReqCycle ReqIF import connector.

8.2 Creation of additional OpenETCS requirements

ReqCycle provides a "local" connector to create new requirements and those requirements can be linked to existing other requirements through different kinds of links (derivation, refinement, decomposition). Those link types have to be defined first in ReqCycle configuration.

To complete

8.3 Creation of links between SysML model elements and requirements

Same solution than in second solution. see 7.3

8.4 Creation of links between SCADE model and requirements

Same solution than in second solution. see 8.4

8.5 Aggregation of traceability links and export

Same solution than in second solution. see 8.5

9 Tool evaluations

Will be done in a separate document.

9.0.1 TODO ReqCycle Evaluation

9.0.2 TODO Reqtify Evaluation

References

[1] Izaskun de la Torre. *openETCS Quality Assurance Plan*, 2014. https://github.com/openETCS/governance/blob/master/QA%20Plan/D1.3.1_QA_Plan.pdf.

[2] Hardi Hungar. *D2.3a openETCS Process*, 2.a2 edition, September 2015. https://github.com/openETCS/requirements/blob/master/D2.3/D2_3a_02.pdf.

[3] openETCS WP3. *Architecture and Design Specification: Software Architecture and external interfaces specification*, 2015. https://github.com/openETCS/modeling/blob/master/openETCS%20ArchitectureAndDesign/D3.5.0/D3_5_0.pdf.

[4] openETCS WP3. *Architecture and Design Specification: Software component design and internal interfaces*, 2015. https://github.com/openETCS/modeling/blob/master/openETCS%20ArchitectureAndDesign/D3.5.3/D3_5_3.pdf.

[5] openETCS WP3. *openETCS Hazard and Risk Analysis and Safety Case Methodology*, 2015. https://github.com/openETCS/validation/blob/master/Reports/D4.2/D.4.2.3-VV-Hazard-and-Risk/OpenETCS_D-4-2-3_Hazard-and-Risk-Analysis-Methodology.pdf.