

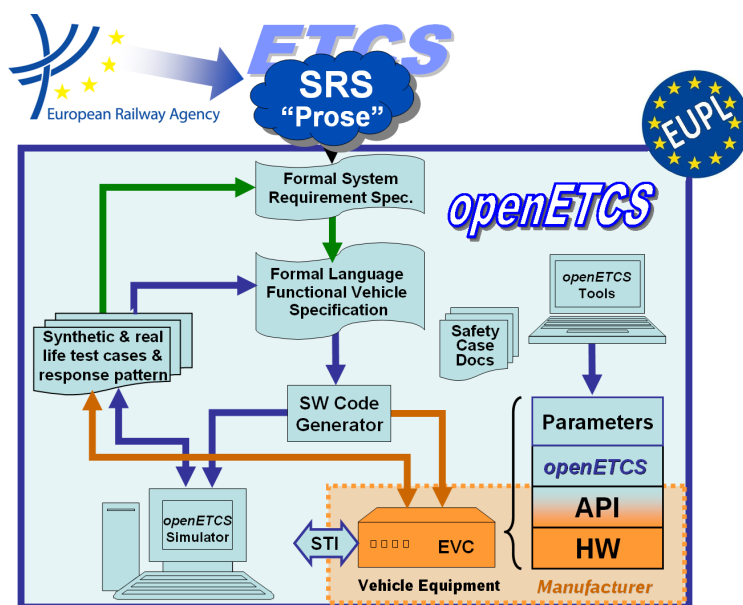
Work-Package 7: "Toolchain"

Traceability Architecture in OpenETCS

WP7 Proposition

Cecile Braunstein, Moritz Dorka, David Mentré and Raphaël Faudou

October 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
October 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	signature	signature	signature
Cécile Braunstein (University Bremen)	()	()	()

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.

Disclaimer: This work is licensed under the "openETCS Open License Terms" (oOLT) dual Licensing: European Union Public Licence (EURL v.1.1+) AND Creative Commons Attribution-ShareAlike 3.0 – (cc by-sa 3.0)

THE WORK IS PROVIDED UNDER openETCS OPEN LICENSE TERMS (oOLT) WHICH IS A DUAL LICENSE AGREEMENT INCLUDING THE TERMS OF THE EUROPEAN UNION PUBLIC LICENSE (VERSION 1.1 OR ANY LATER VERSION) AND THE TERMS OF THE CREATIVE COMMONS PUBLIC LICENSE ("CCPL"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS OLT LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

<http://creativecommons.org/licenses/by-sa/3.0/>
<http://joinup.ec.europa.eu/software/page/eupl/licence-eupl>

Table of Contents

Document Information	iv
1 OpenETCS traceability scope	1
1.1 OpenETCS requirements with regards to standard engineering levels	1
1.2 OpenETCS models with regards to requirement engineering levels	2
2 OpenETCS traceability priorities and expectations	4
2.1 Traceability priorities	4
2.2 Traceability expectations	5
2.2.1 Traceability Process	5
2.2.2 OpenETCS traceability main scenario	5
3 OpenETCS tool chain requirements about traceability	7
3.1 Tool chain capabilities to support requirement management	7
3.2 Tool chain capabilities to support requirement traceability	8
4 Tool chain architecture to support traceability	10
4.1 Existing tooling	10
4.2 First Solution	10
4.2.1 TODO ReqCycle Evaluation	11
4.2.2 TODO Reqify Evaluation	11
5 Activities Details	12
5.1 Subset-026 import	12
5.1.1 Script description	12
5.1.2 Unique ID definition	12
5.2 Requirement Database	13
5.3 Requirement management	13
5.4 System Model	13
5.5 Interface Definition	14
5.6 Functional model	14
5.7 TODO Verification and Validation	14

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 OpenETCS traceability scope

Requirements traceability concerns relations between requirements existing at different engineering levels and relations between requirements and other engineering artefacts (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 the requirements and of the models that we want to trace in openETCS project. Next paragraphs recall the different engineering levels defined by ISO 15288:2015 (standard concerning systems life cycle) and position OpenETCS requirements, models and documents with regards to those engineering levels.

1.1 OpenETCS requirements with regards to standard engineering levels

ISO 15288:2015 defines 3 main generic levels:

- problem definition,
- system/Sub system definition (recursive decomposition)
- physical building blocks of software, hardware (mechanical, electrical, electronics, plastic...) or procedures that can be realized independently and integrated to provide whole system

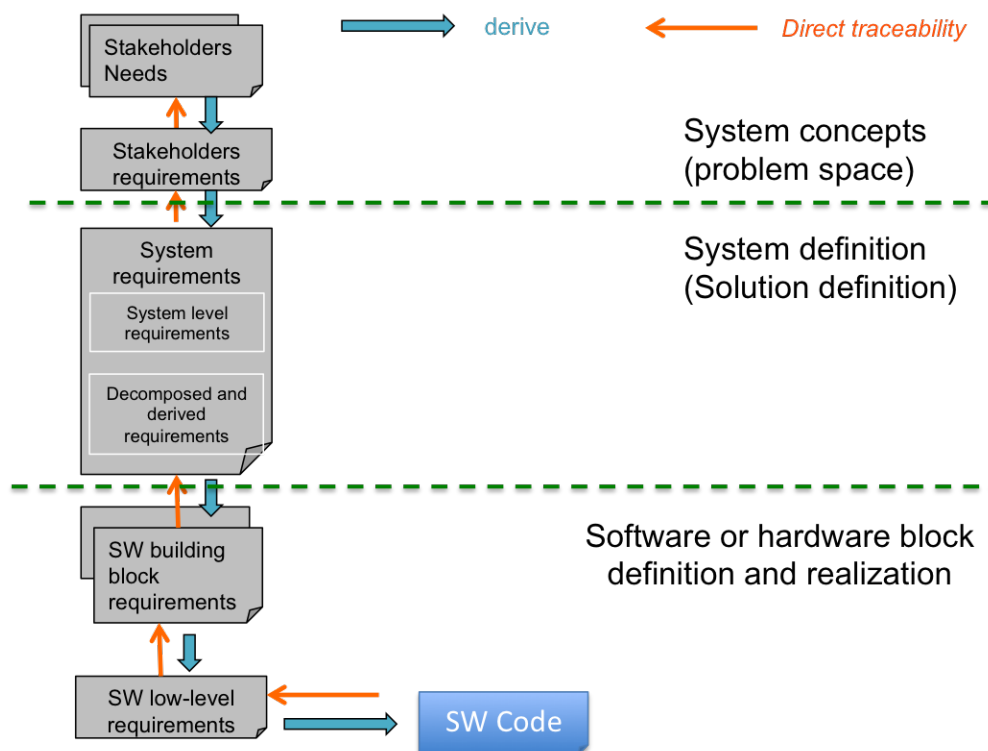


Figure 1. ISO 15288:2015 standard Systems Engineering requirement levels

Concerning OpenETCS project, there are 5 main sources for requirements:

- User stories, concerning problem definition
- CENELEC regulation requirements that are part of stakeholder requirements
- SRS Subset 026 that can be considered as system level requirements
- openETCS new requirements created during system definition by decomposition or derivation of SRS Subset 026 requirements
- openETCS API, that provides software requirements to integrate a vendor specific platform

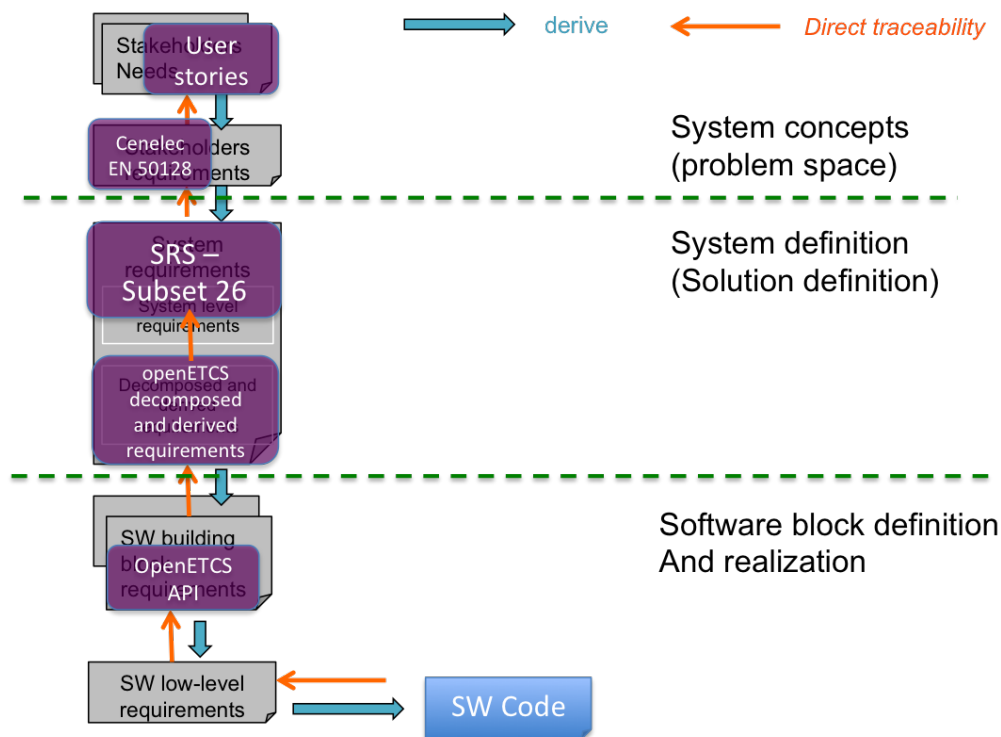


Figure 2. OpenETCS requirements with regards to standard engineering levels

1.2 OpenETCS models with regards to requirement engineering levels

Models can be used for different purposes including illustration of concepts, operational scenarios, formalization of requirements, product breakdown structure... and can have different scopes. Figure 3 shows a possible classification of models aligned on the standard requirement engineering levels introduced in previous paragraph.

Figure 4 illustrates the different models produced in the openETCS project and positionned on standard engineering levels:

- User stories model that refines user stories
- OpenETCS EVC model, that describes system breakdown structure with system elements (including ETCS OnBoard Unit) and their interfaces
- Data dictionary model, that stores all messages, data and functions used in architecture interface definitions
- OpenETCS Onboard Unit functions model, a formal executable model that focuses on ETCS Kernel functions addressed in Subset026 SRS

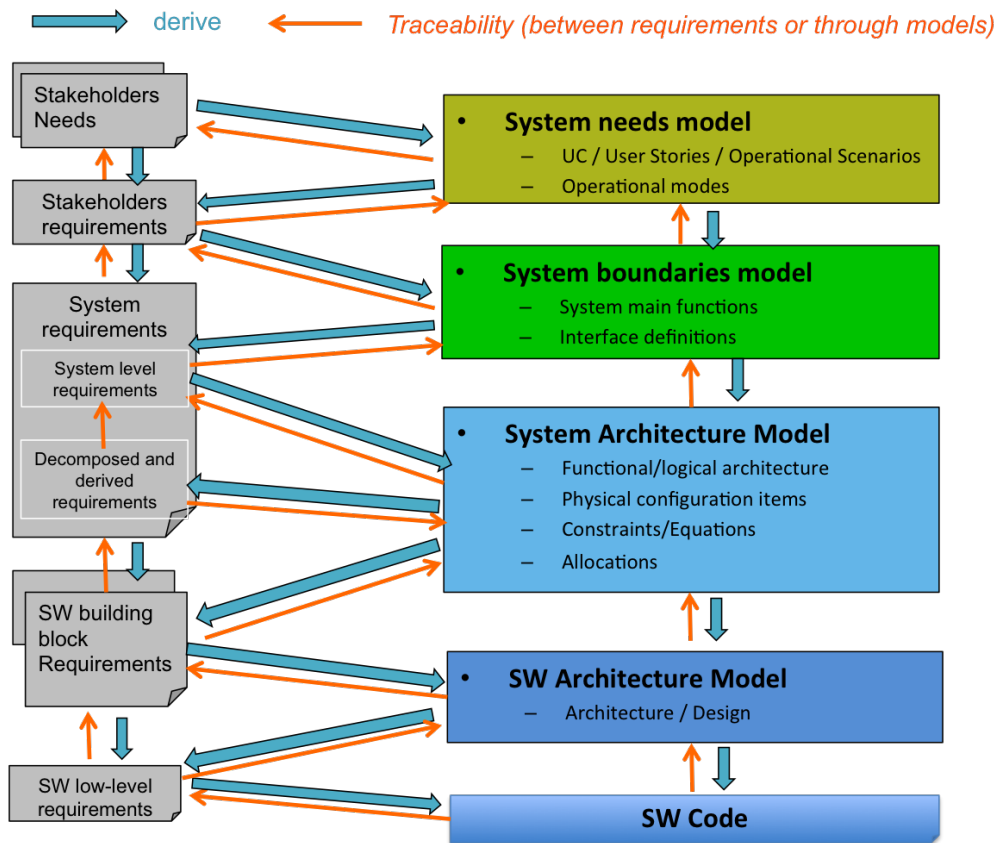


Figure 3. Possible classification of models with respect to standard requirement engineering levels

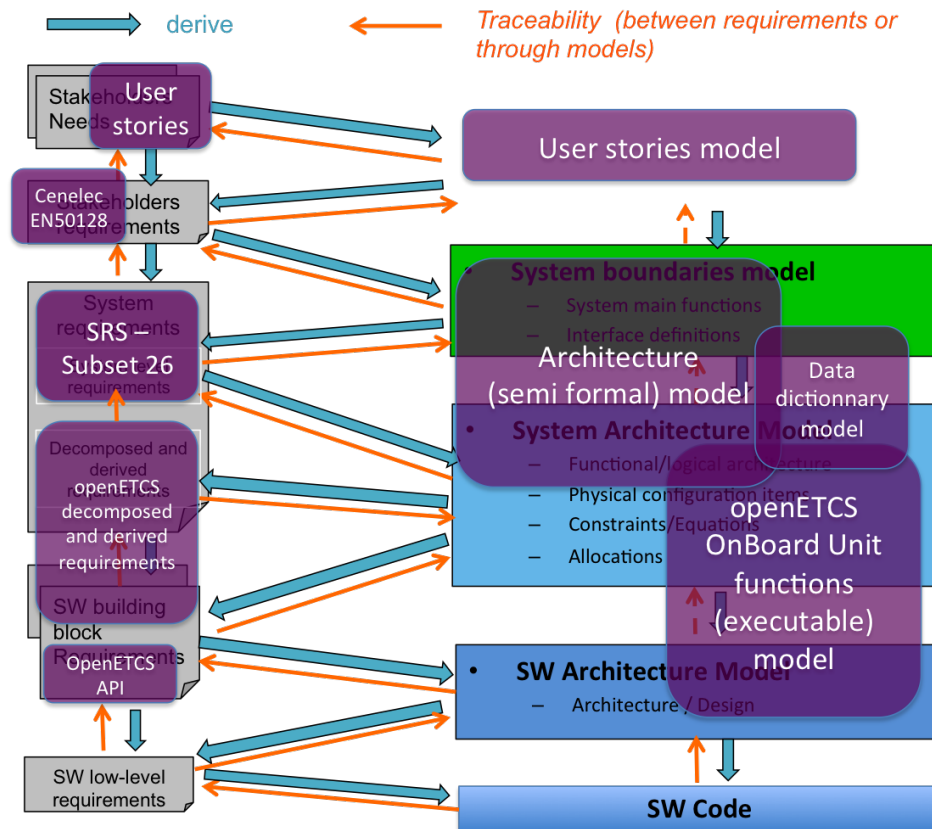


Figure 4. OpenETCS requirements and models with respect to standard requirement engineering levels

2 OpenETCS traceability priorities and expectations

2.1 Traceability priorities

Most important traceability chain concerns links between ETCS OBU functions model and Subset026 SRS requirements and all other requirements created during system analysis, either by decomposition, refinement or derivation. ETCS OBU functional model is detailed down to software 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.

Other "nice to have" traceability chains concern system model and data dictionary model. Those models need to be traced to Subset026 SRS requirements in order to ensure that they are representative of OpenETCS system and are complete (all requirements are covered, no missing requirement).

Figure 5 highlights traceability chains with highest priority (arrows with largest size).

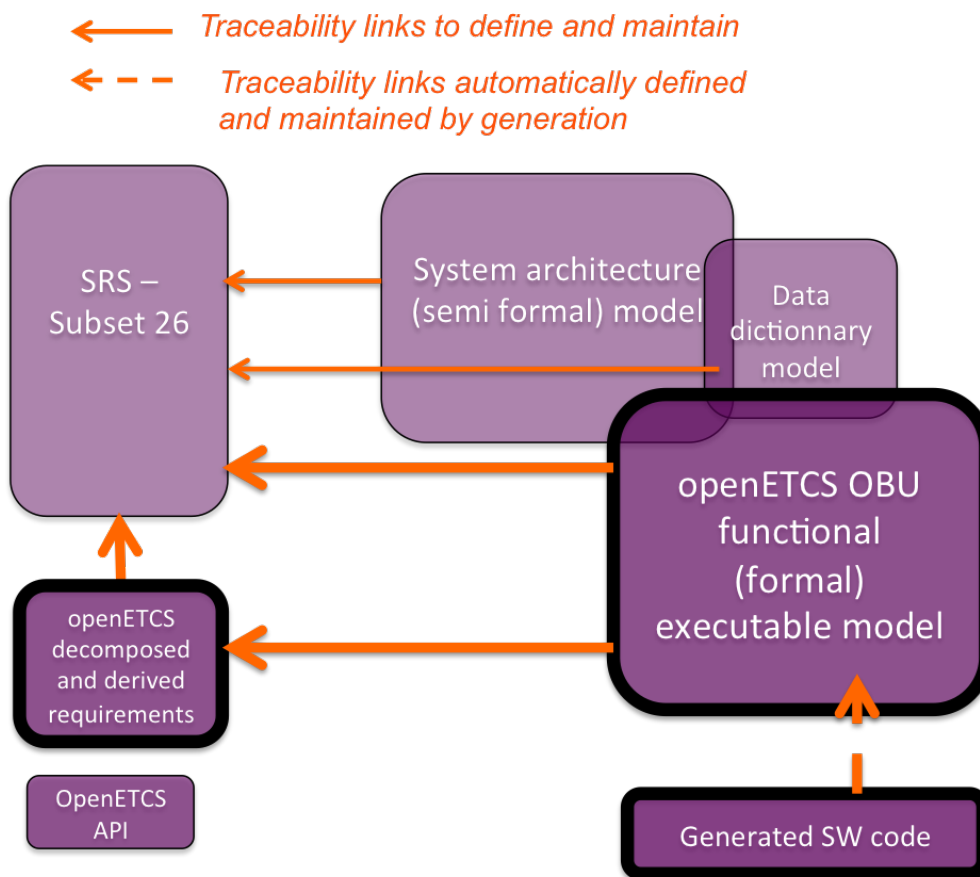


Figure 5. OpenETCS traceability chains with highest priority

2.2 Traceability expectations

2.2.1 Traceability Process

From the Subset026 SRS specification we want to be able to trace all artifacts that implement it.

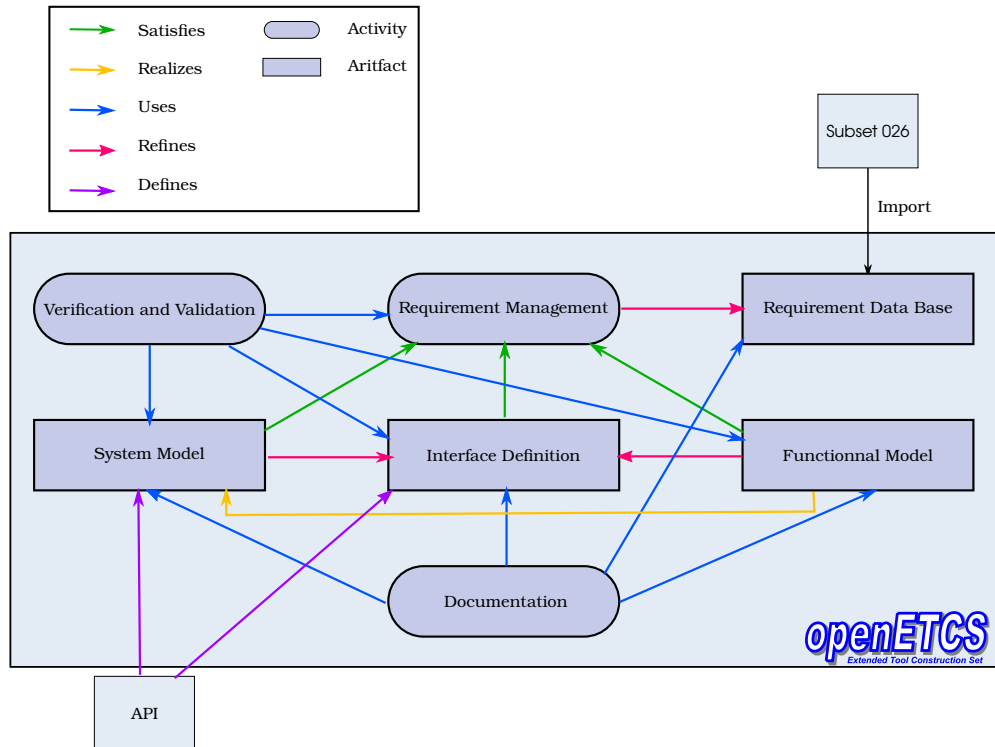


Figure 6. Traceability architecture between artifacts

Figure 6 highlights the different artifacts or activities and their links. Traceability activities should allow to trace a requirement within all the artifacts and/or activities that are using it.

The goal of traceability is that, from any artifacts produced, we are able to track which requirement it realizes, implements or refers to. These links can take different form and be done by different tools.

2.2.2 OpenETCS traceability main scenario

Here is a global scenario that illustrates the way an OpenETCS system designer will manage existing reference requirements and will create new ones by decomposition or derivation to a lower engineering level, how he/she will provide a functional solution (functional model) to satisfy the different OpenETCS requirements and verification means (tests or else) to verify them.

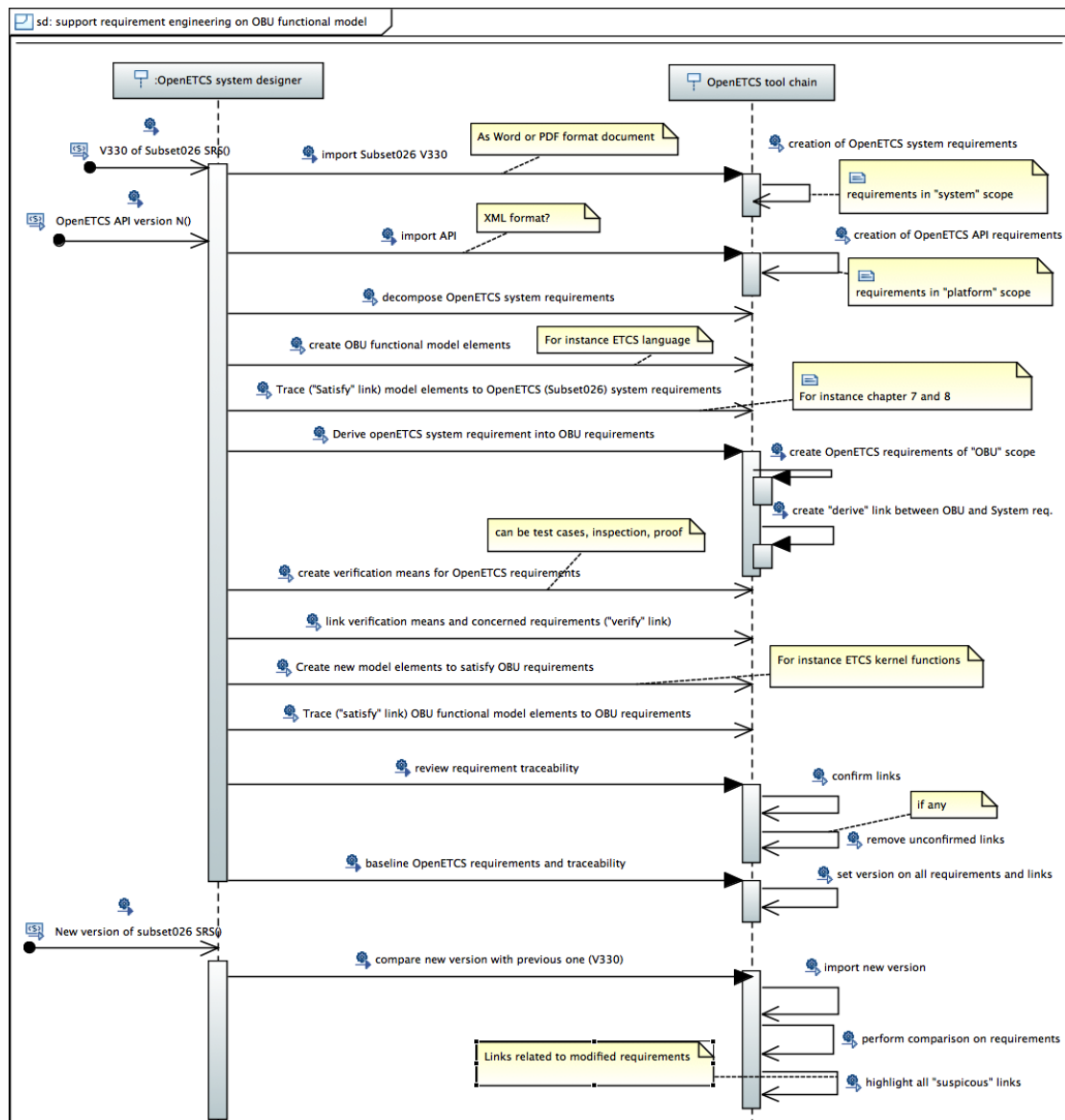


Figure 7. Main traceability scenario concerning tool chain

3 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.

3.1 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 subset026 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.

3.2 Tool chain capabilities to support requirement traceability

Requirement traceability activity consists in ensuring that all product engineering artefacts (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 artefact, based on a given link template/type (refine, derive, implement, verify...).
 - **OpenETCS example:** create a "Refine" link between one Subset026 SRS requirement and one OnBoard Unit function, with "status" link attribute set to "defined".
- 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

4 Tool chain architecture to support traceability

Traceability architecture is mainly based on two components: requirement management component and requirement traceability component.

4.1 Existing tooling

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:

- 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.
- 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, 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.

The present solution proposes a first "by hand" approach, to deal with traceability without having all the proper tools already integrated into the tool chain.

4.2 First Solution

TO COMPLETE...

Figure 8 proposes a first solution to handle traceability. The basic idea is to always refer to the same 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. But it may be possible for requirement management tools to refine these requirements but they should guarantee the following rules:

1. The requirement's structure is not modified.
2. No requirement should be deleted.
3. Newly added requirements should be either a refinement or a decomposition of an existing one.

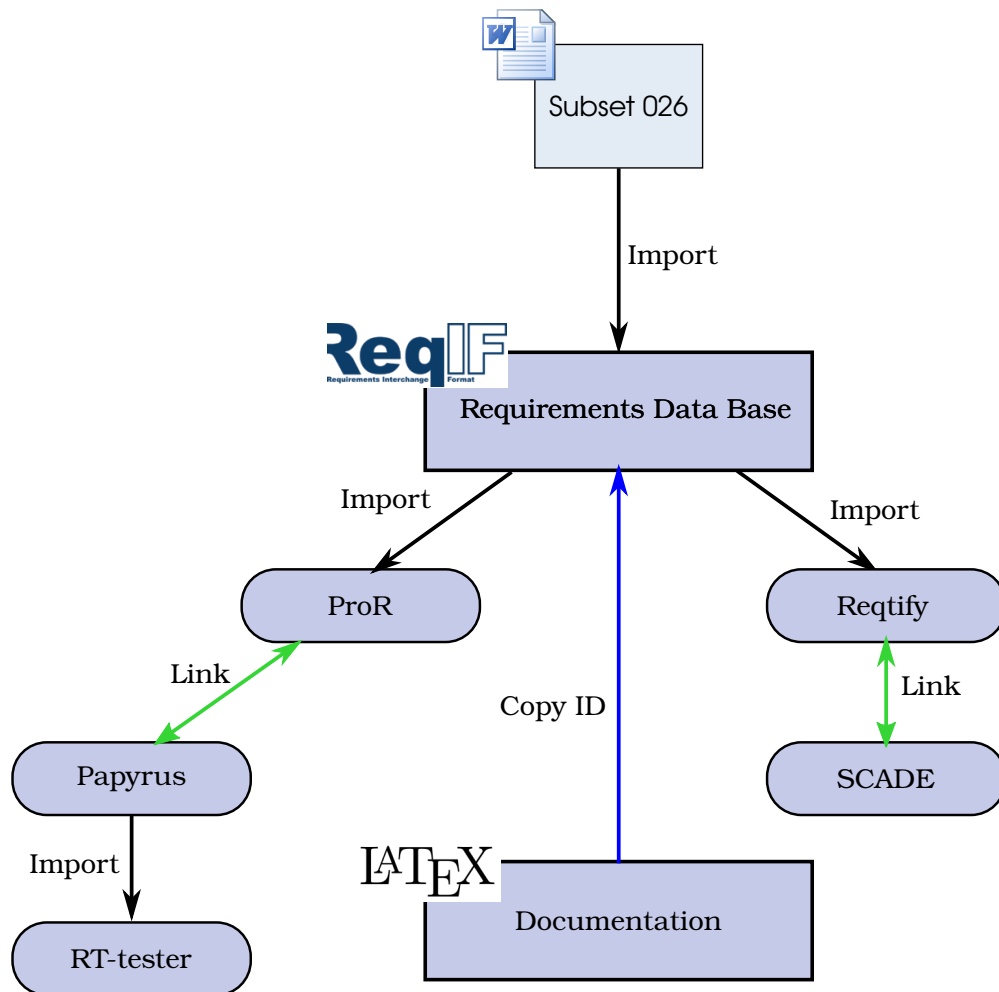


Figure 8. Traceability architecture first solution

4. The identifier pattern should be respected.
5. Identifiers should stay unique.

4.2.1 TODO ReqCycle Evaluation

4.2.2 TODO Reqtify Evaluation

5 Activities Details

5.1 Subset-026 import

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

5.1.1 Script description

The script
needs a name

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.

5.1.2 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 9. 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]

5.2 Requirement Database

The requirement data base is a ReqIF file-based.

5.3 Requirement management

The requirement management allows us to have a look at the set of requirements and performs some actions. The set of requirements can be automatically imported from a Req-IF file.

The requirement management should be use to refine the high-level requirements from the import. It can be use to give more precise information about :

- Which requirements are implementable
- Who is assigned to implement a set of requirements
- How and if the requirement is decomposed.
- TODO (list to be refined)

In the OpenETCS tool chain this tasks is done with ProR. Alternatively to deal with the closed source path the requirement management may be done by Reqtify Gateway of SCADE.

5.4 System Model

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

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.

5.5 Interface Definition

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 refines by the modeling team without changing the existing implemented requirements.

5.6 Functional model

The function model is the implementation of the system defines in the system model. It should covers the set of implementable requirements.

5.7 TODO Verification and Validation