OETCS/WP7/07.3.5

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

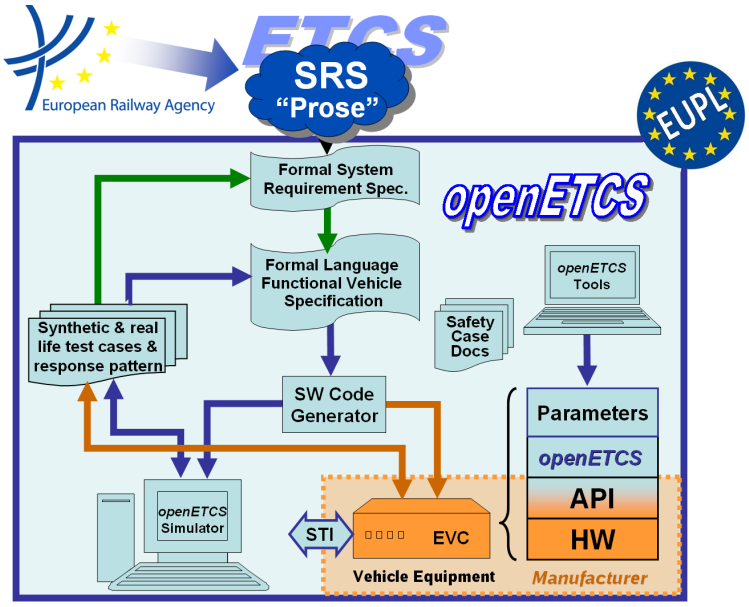
Work-Package 7: “Toolchain”

# Traceability Architecture in OpenETCS

**WP7 Proposition**

Cecile Braunstein, Moritz Dorka, David Mentré and Raphaël Faudou

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<br><br>Cécile Braunstein<br>(University Bremen) | signature<br><br>() | signature<br><br>() | signature<br><br>() |

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 propostion 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 | Update concerning model scope |

# 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)
- building blocks of software, hardware (mechanical, electrical, electronics, plastic...) or procedures that can be realized independently

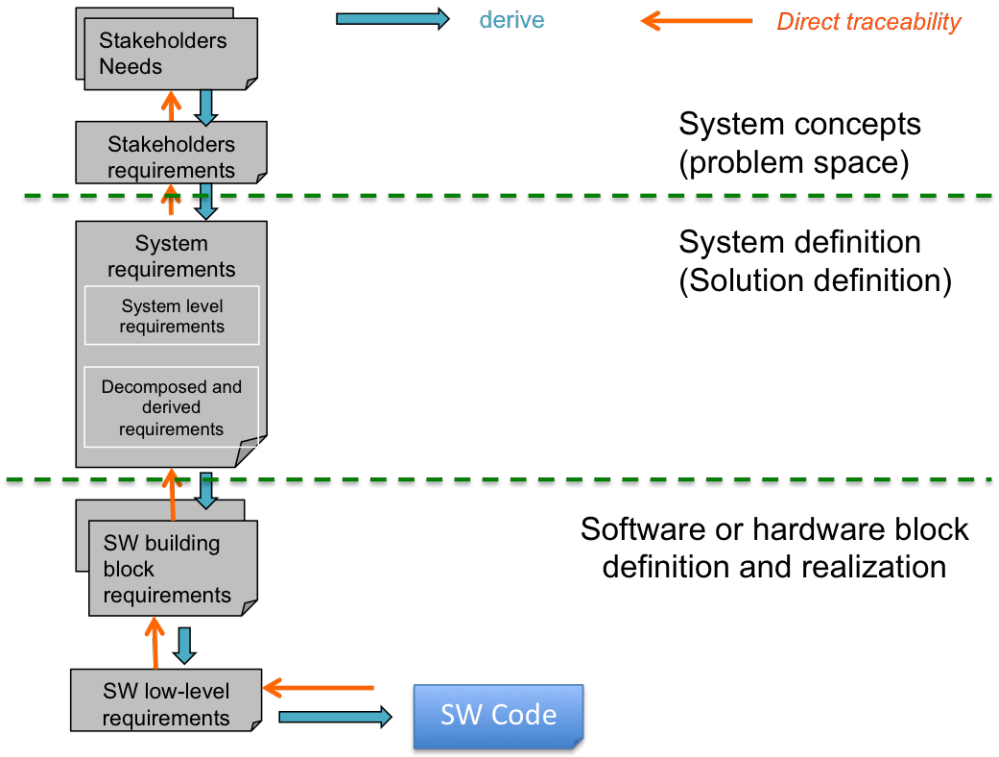



**Figure 1. IS0 15288:2015 standard Systems Engineering requirement levels**

Concerning OpenETCS project, there are 4 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

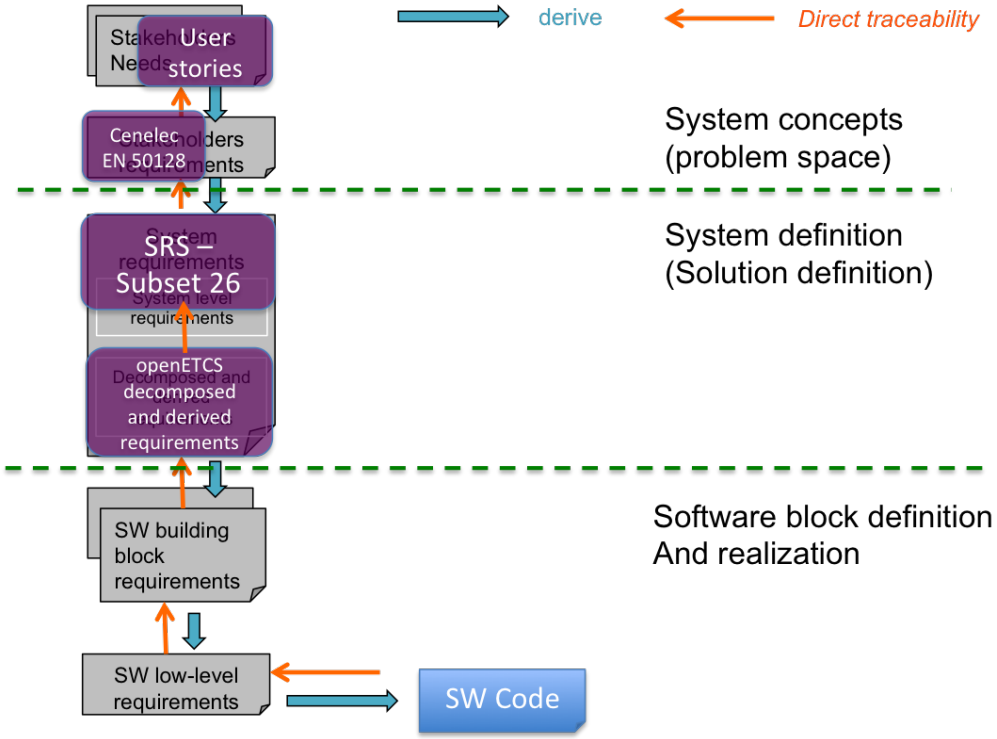



**Figure 2. OpenETCS requirements with regards to standard engineering levels**

## 1.2 OpenETCS models with regards to requirement engineering levels

Models can be used for different purposes including illustration of concepts, 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.

If we look at the different models produced in the openETCS project we find:

- User stories model that refines user stories
- OpenETCS architecture model, that describes high level system breakdown structure with system elements 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 chapters 7 and 8 of Subset026 SRS (ETCS Language and messages)

In order to depict their scope and relative positions we can try to align them on standard engineering levels presented previously. Figure 4 illustrates those positions.

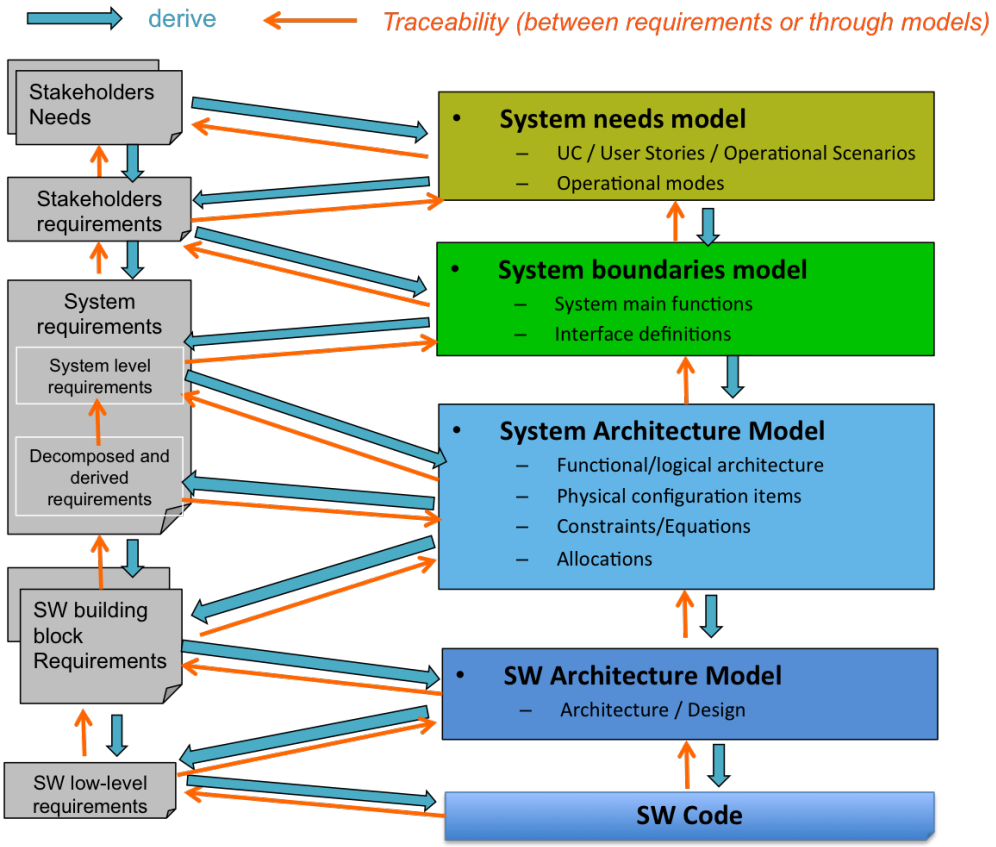



**Figure 3. Possible classification of models with respect to standard requirement engineering levels**

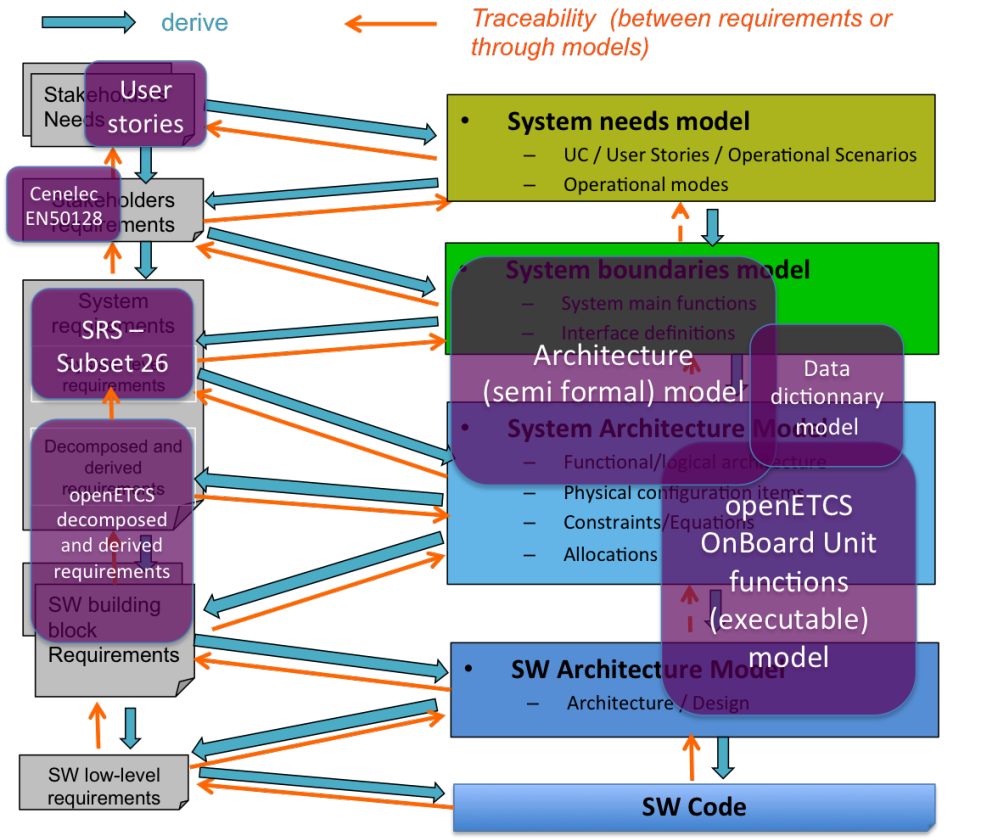



**Figure 4. OpenETCS requirements and models with respect to standard requirement engineering levels**

# 2 OpenETCS traceability priorities

Most important traceability chain concerns links between ETCS OBU functions model and Subset026 SRS requirements, especially chapter 7 and 8, and all other requirements created during system analysis, either by decomposition or by derivation to system elements. 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 (meets) 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/architecture model and data dictionary (interface definition) model. Those models need to be traced to Subset026 SRS requirements in order to ensure that they are representative of OpenETCS system and complete (all requirements are covered, no missing requirement). Another reason concerns impact analysis: with traceability links, it becomes possible to perform impact analysis through models when system requirements change (just have to follow traceability links to find associated model elements).

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

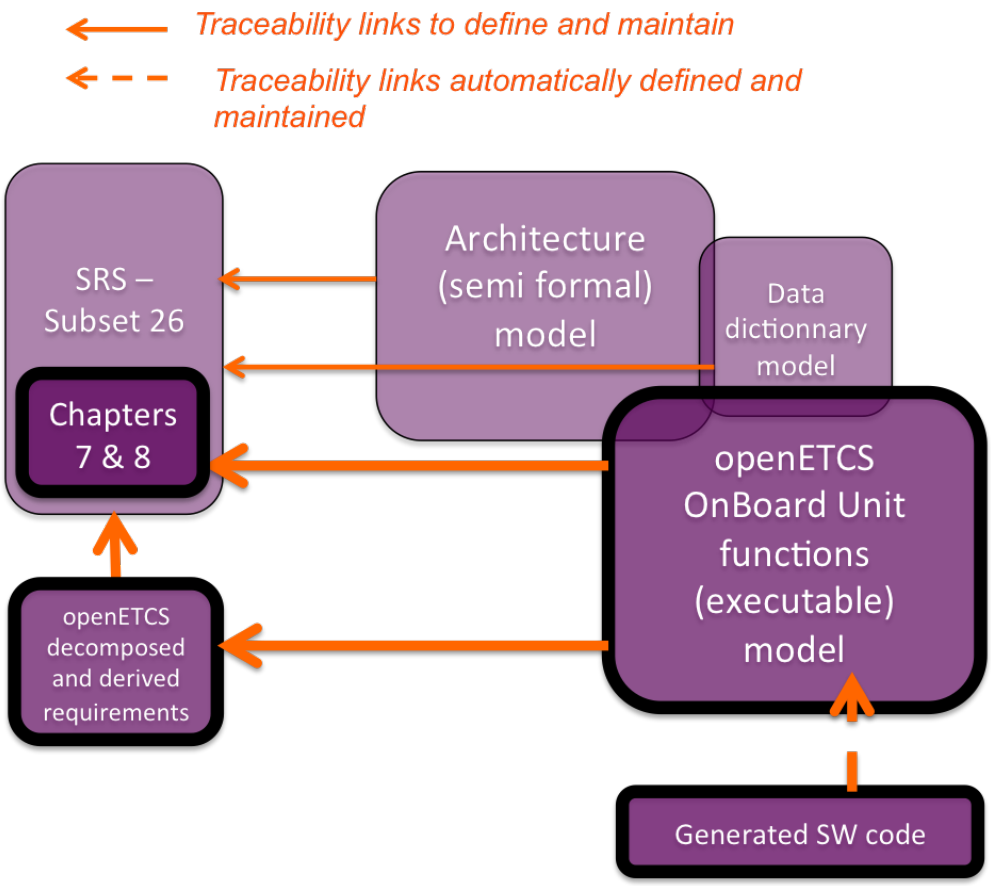



**Figure 5. OpenETCS traceability chains with highest priority**

# 3 OpenETCS tool chain Proposition

## 3.1 Traceability Process

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

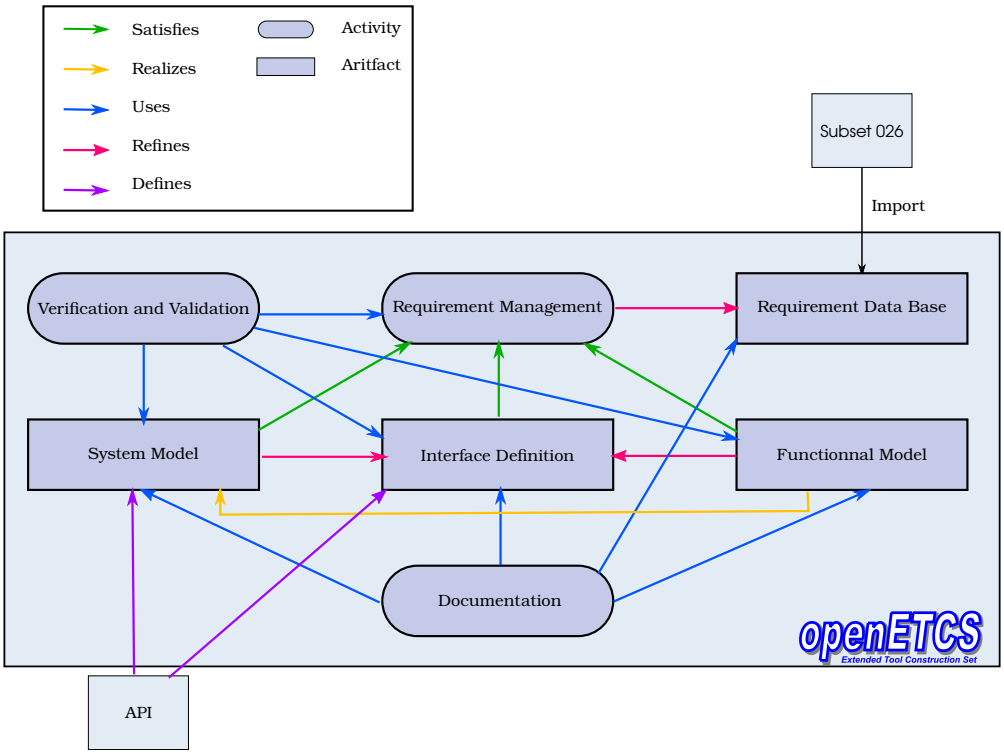



**Figure 6. Traceability architecture between artifacts**

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. Two traceability tools will be evaluated before being integrated in the OpenETCS tool chain: ReqCycle and Reqtify.

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.

## 3.2 First Solution

Figure 7 proposes a first solution to handle traceability. The basic idea is to allways 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

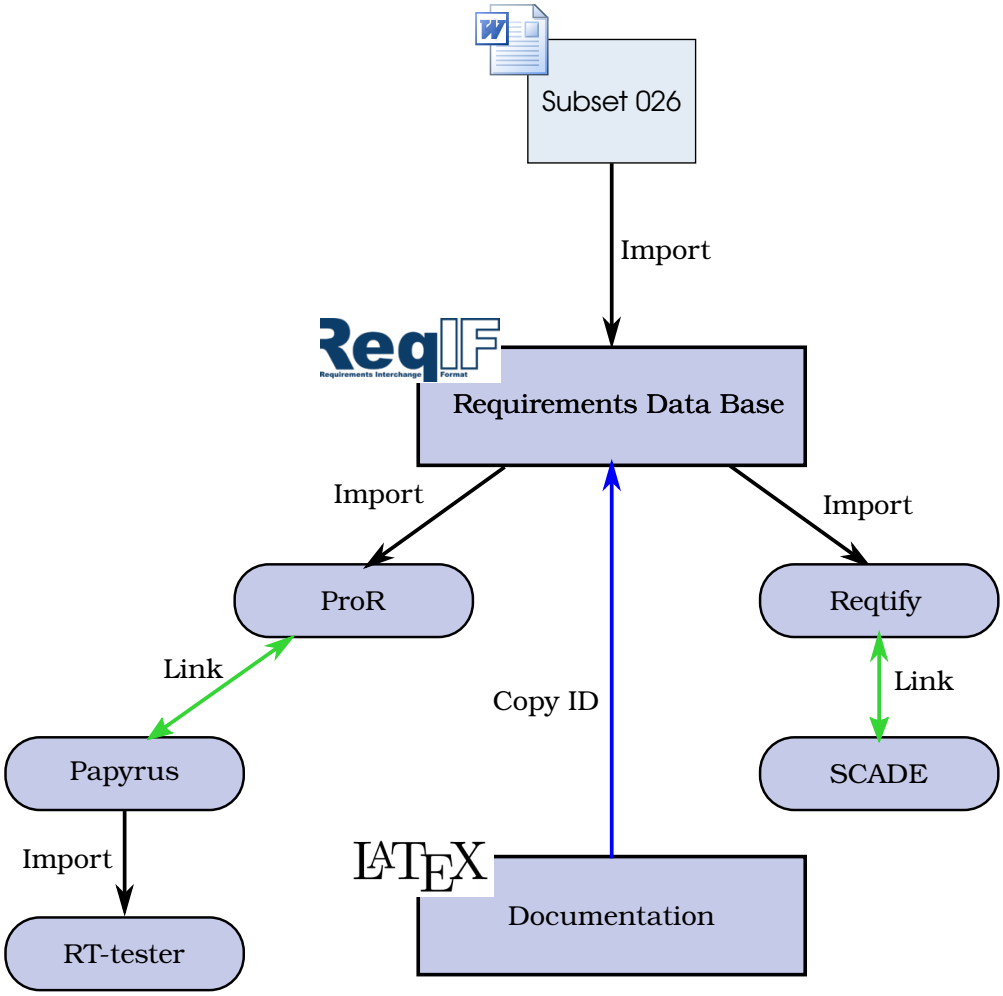



**Figure 7. Traceability architecture first solution**

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

### 3.2.1 TODO ReqCycle Evaluation

### 3.2.2 TODO Reqtify Evaluation

# 4 Activities Details

## 4.1 Subset-026 import

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

### 4.1.1 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

### 4.1.2 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 8. 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]`

## 4.2 Requirement Database

The requirement data base is a ReqIF file-based.

## 4.3 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.

## 4.4 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.

## 4.5 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.

## 4.6 Functional model

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

## 4.7 TODO Verification and Validation