OETCS/WP7/07.3.5

openETCS

ITEA2 Project
Call 6 11025
2012 – 2015

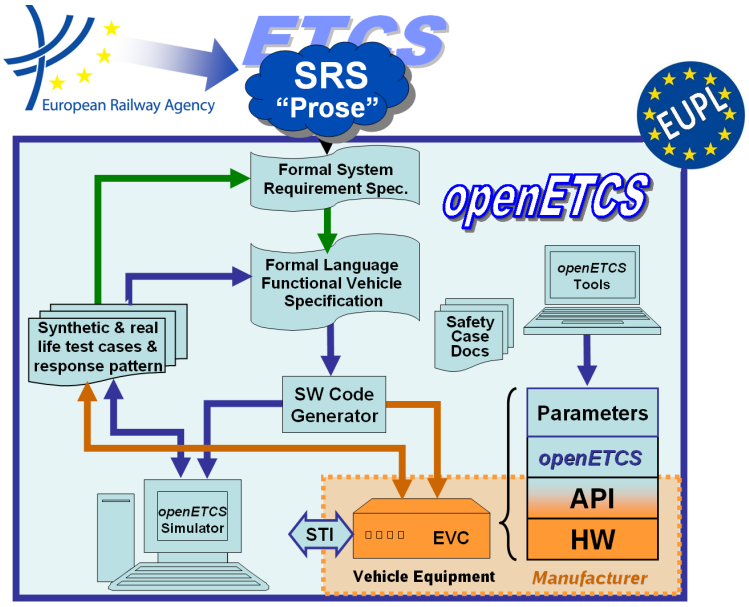
Work-Package 7: “Toolchain”

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 ..... **v**

1 Introduction **1**

1.1 Document purpose 1
1.2 Document scope 1
1.3 Document organization 1

2 OpenETCS traceability context **2**

2.1 Recall of OpenETCS target vision and development process 2
2.2 OpenETCS project context-of-interest 2
2.3 OpenETCS design process (WP3) and design artifacts 3
2.4 OpenETCS VV and safety process (WP4) 6

3 OpenETCS tool chain general requirements about traceability **7**

3.1 OpenETCS high level requirements about traceability support by tool chain 7
3.1.1 Which model? 7
3.1.2 What does "tracing model" mean? 8
3.2 Tool chain capabilities to support requirement management 8
3.3 Tool chain capabilities to support requirement traceability 10

4 Short term development priorities and associated practical scenarios concerning traceability **11**

4.1 Main design priorities and current requirement organization 11
4.2 OpenETCS traceability main scenarios 13
4.2.1 P1 Software specification Phase 13
4.2.2 P2 - Software Architecture Modeling Phase 15
4.2.3 P3 - Software Behavior Modeling and Integration Phase 15
4.2.4 P5 - Formal Validation Phase 17

5 Tool chain current physical architecture solution to support traceability **18**

5.1 Overview of current traceability architecture solution 18
5.2 P1 phase: ProR, Scade System, Papyrus and ReqCycle 19
5.2.1 Import of reference requirements through a Script 19
5.2.2 Classification of reference requirements through ProR 19
5.2.3 Creation of additional requirements through ProR 19
5.2.4 Link requirements and overall test specification with ReqCycle 20
5.2.5 Update of SRS subset26 20
5.2.6 technical challenges 21
5.3 P2 phase: ProR, Scade System, Papyrus and ReqCycle 22
5.3.1 verification of SW architecture through ReqCycle 22
5.3.2 Creation of traceability links between a requirement and a Papyrus SysML with ReqCycle 23
5.4 P3 phase: ProR, Scade suite and ??? 23
5.5 P4 phase: SCADE suite and Scade KCG 25
5.6 P5 phase: Verification and validation of SW product 25

6 Limits of current traceability architecture and axes of investigation for improvement **26**

6.1 Current identified limits 26
6.1.1 Limits concerning the baseline of requirements used as input for traceability 26

6.1.2 Limits concerning additional requirements 26
6.1.3 Limits concerning traceability with SysML elements 26
6.1.4 Limits concerning traceability with SCADE elements 27
6.1.5 Limits concerning management of all traceability links 27
6.2 Investigation axes for improvements 27
6.2.1 Ability to export Scade model as Eclipse EMF model 27
6.2.2 Combined solution for requirement management and traceability tools 27
6.2.3 ReqCycle used as centralized solution and ProR for requirement edition 28

References 30

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].
- OpenETCS higher level requirements (especially concerning traceability) - D2.6.9 - [2].
- Definition of OpenETCS Development Process - D2.3a v02 - [3]
- OpenETCS Architecture and Design Specification - D3.5.0 - [4] and D3.5.1 - [5]
- Safety plan - D4.2.3 - [6]

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, design, VV and Safety processes.

Chapter 3 defines tool chain general requirements concerning traceability driven from [2] and from systems engineering experience.

Chapter 4 refines short term development design process and main priorities to deliver demonstrators for ITEA2 project. It provides detailed practical scenarios and expected support on verification and especially traceability during the different phases of development process.

Chapter 5 presents current traceability architecture defined to support practical scenarios identified in chapter 4 for short term development target. This architecture is based on use of existing tools/features.

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

2 OpenETCS traceability context

Requirement traceability concerns relations between requirements existing at different engineering levels and relations between requirements and design or verification engineering artifacts that can exist through different formats (models, documents, code...).

Before describing the way openETCS tool chain supports creation and maintenace of traceability links, it is important to clearly define the scope of openETCS activity and the artifacts that we want to trace over project life cycle. Next paragraphs recall OpenETCS vision, development process and current scope of development.

2.1 Recall of OpenETCS target vision and development process

[3] 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.

[6] 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.

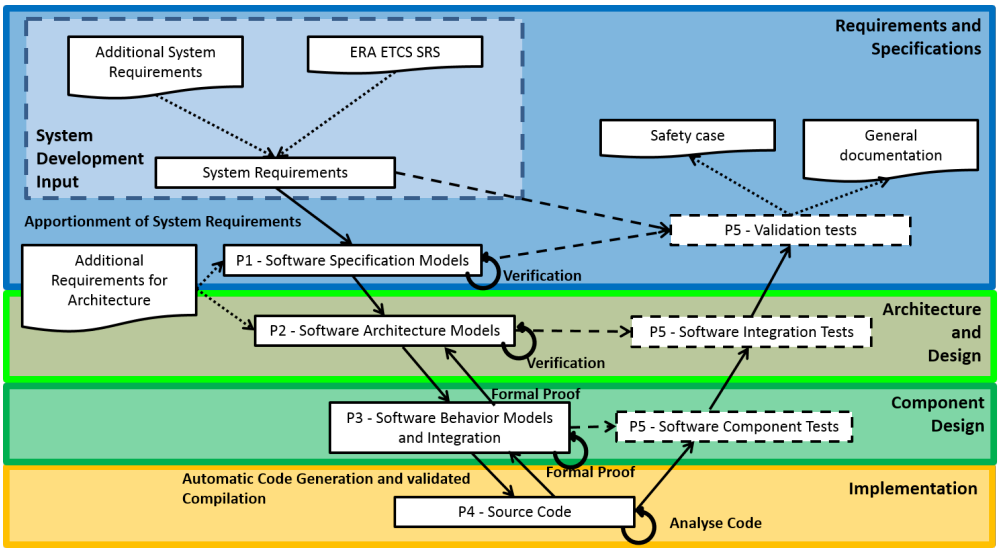


Figure 1. OpenETCS software development process

2.2 OpenETCS project context-of-interest

[5] 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.

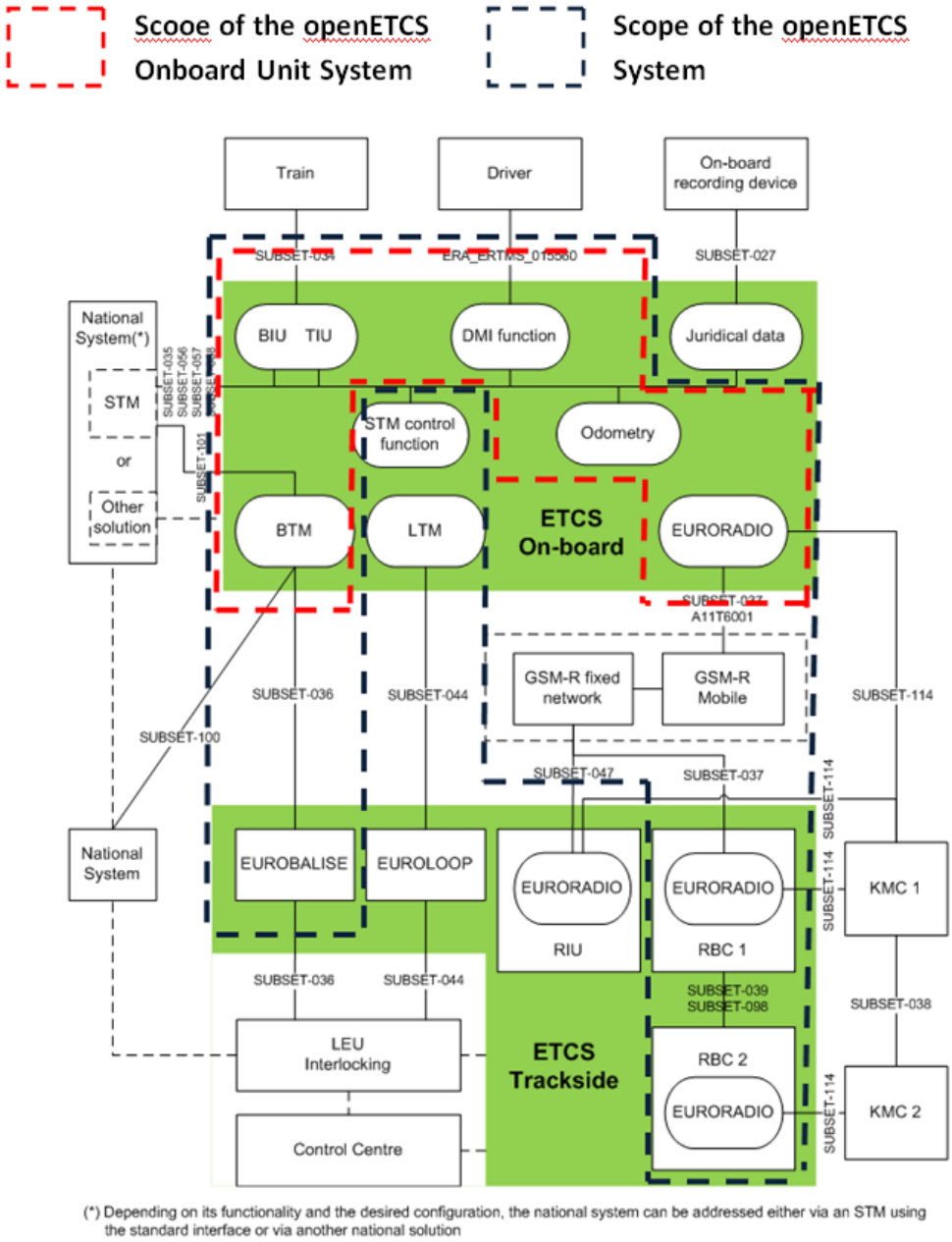


Figure 2. OpenETCS system of interest (dotted black line) to cover demonstration objective of ITEA2 project and according to ERA TSI Chapter 2

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 and focused on the reference track (ETCS L2 Utrecht- Amsterdam).

This approach leads to the following architecture and design artifacts described in [4] :

- 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, mainly data flow)
- OpenETCS application software executable model, a model defined with SCADE tool. This model integrates several behavioral SCADE models that realize functional block interfaces identified in SysML 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.

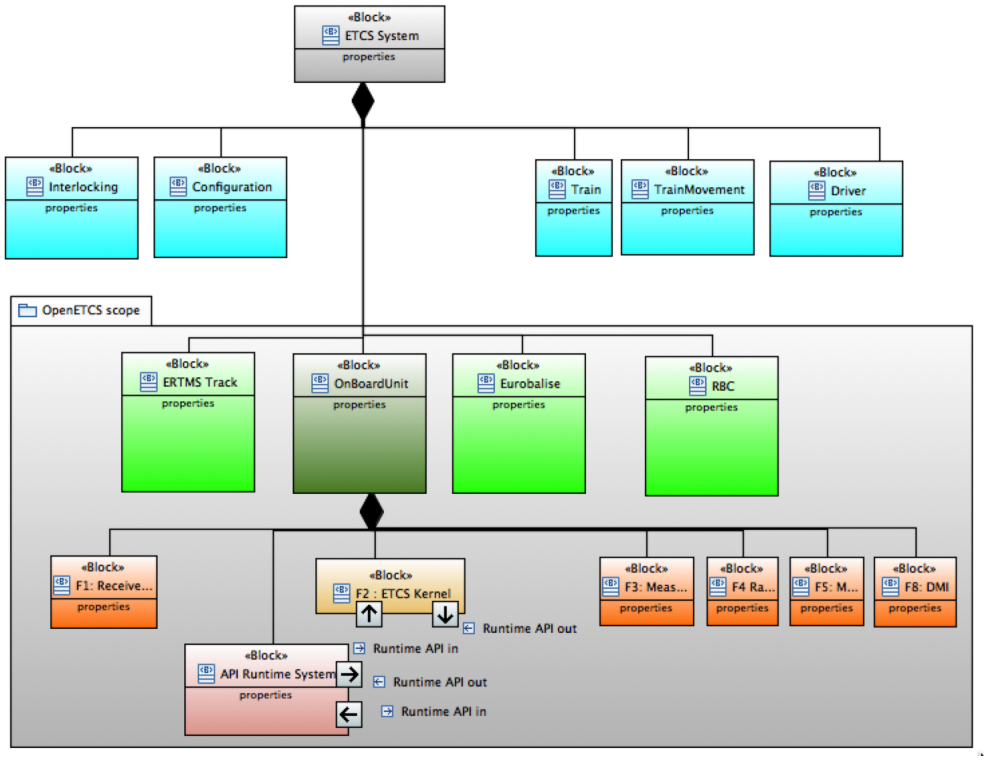


Figure 3. OpenETCS architecture with SysML: from system to software components

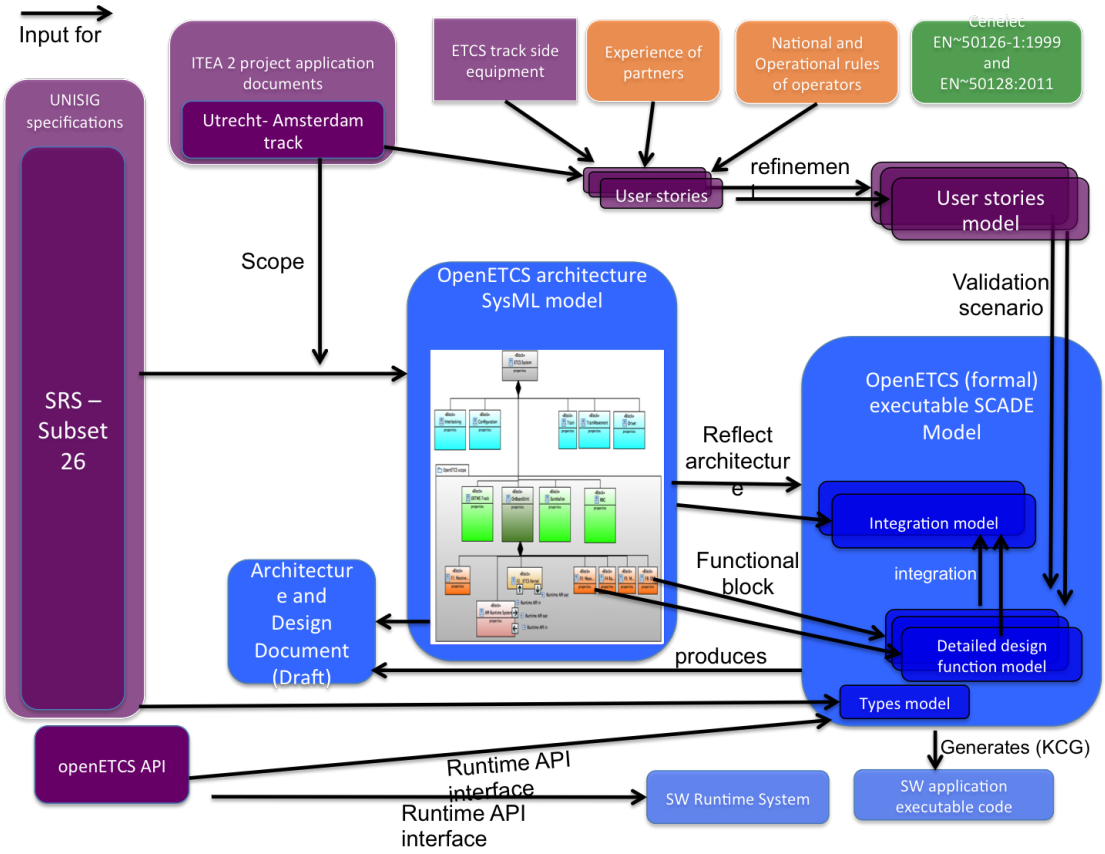


Figure 4. Relations between main WP3 Design Documents and reference input artifacts (SRS, Utrecht-Amsterdam reference track and OpenETCS API)

2.4 OpenETCS VV and safety process (WP4)

[6] 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.

Concerning application of safety process in the scope of openETCS ITEA project, it is not possible to apply it on full scope of the project with all requirements expressed in CENELEC engineering standards . So a proof of concept has been defined in order to apply it with a model based approach. PoC is based on Management of Radio Communication (MoRC). Results are available here: https://github.com/openETCS/validation/tree/master/VnVUserStories/VnVUserStoryAll4Tec-AEbt but safety requirements have not been integrated into the project requirements database (.ReqIF files).

But this PoC has demonstrated feasibility and usefulness but has not been extended to the full scope of project.

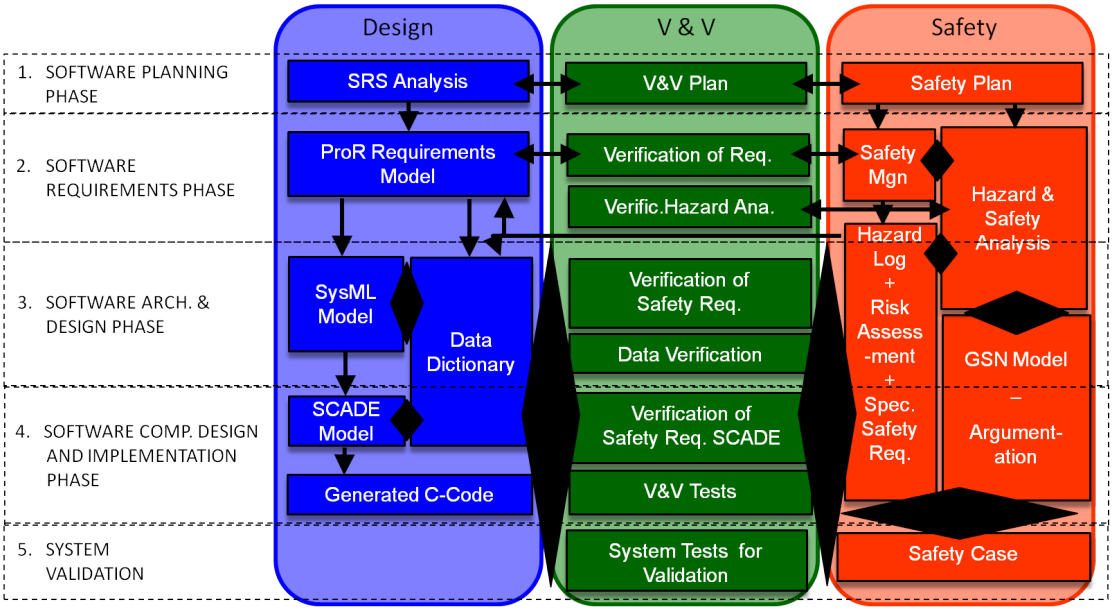


Figure 5. OpenETCS development relations between design, verification and validation and safety activities

3 OpenETCS tool chain general requirements about traceability

This chapter starts by recalling openETCS tool chain high level requirements concerning traceability. They have high priority as reference inputs for the project.

But we will see that those requirements are hard to apply "as-is" because they address generic artifacts with undefined scope and poorly defined semantics. So we discuss them and then complete them with more practical tool chain capabilities derived from systems engineering experience about traceability and first expectations already captured from introduction.

3.1 OpenETCS high level requirements about traceability support by tool chain

[2] defines a set of high level requirements for the project including some concerning tool chain. Among them we find R-WP2/D2.6-02-079 that addresses traceability support through a set of sub requirements. We recall them below.

R-WP2/D2.6-02-079 The tool chain shall allow traceability between:

- R-WP2/D2.6-02-079.01 the documentation/requirements and the models,
- R-WP2/D2.6-02-079.02 the documentation/requirements and the tests,
- R-WP2/D2.6-02-079.03 the models and the tests,
- R-WP2/D2.6-02-079.04 each required level of implementation and the lower level (e.g. higher level model vs lower level model, lower level model vs source code. . .),
- R-WP2/D2.6-02-079.05 the documentation/requirements and the models,
- R-WP2/D2.6-02-079.06 the documentation/requirements and the safety properties/requirements,
- R-WP2/D2.6-02-079.07 the models and the safety properties/requirements,
- R-WP2/D2.6-02-079.08 the tests and the safety properties/requirements.

3.1.1 Which model?

A model, even if we restrict it to the context of systems engineering, may have a lot of purposes including formalization of requirements, architecture definition, property verification or simply illustration of concepts for communication.

Figure 6 suggests a possible typology of systems engineering models that illustrates the wide range of model purposes.

According to its purpose and intended use in the development process, a model can be considered as a reference engineering artifact maintained in configuration or as an intermediate means that helped building some engineering artifact but that can be thrown away after some time.

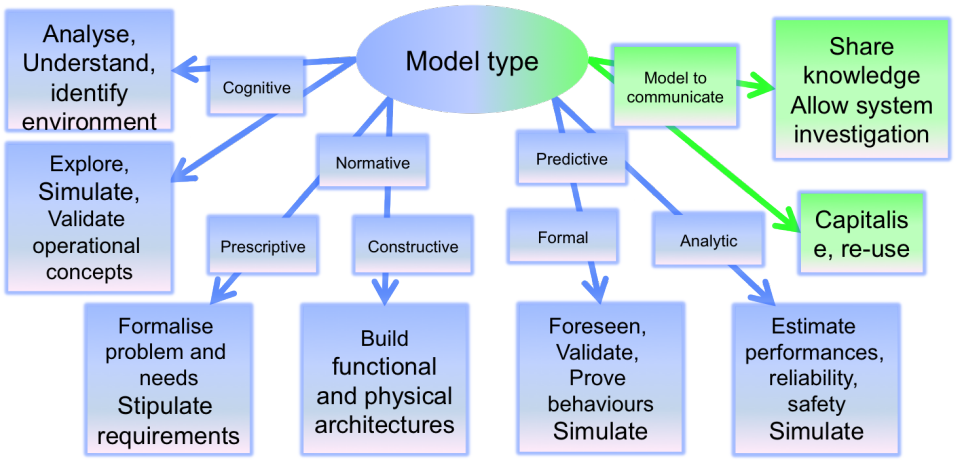


Figure 6. Possible typology of systems engineering models

As creation (and maintenance over time) of traceability links is time and efforts consuming and we suggest restricting it to models that are or contain key engineering artifacts (configuration items).

With that precision, requirement D2.6-02-079.01 would then become: "the tool chain shall allow traceability between documentation/requirements and the models that are or contain configuration items"

3.1.2 What does "tracing model" mean?

Tracing a model is generally not at the right scope. More precisely we want to trace artifacts contained in a model (like block properties, state transitions, data flow...). So we need to be able to identify those artifacts in the model and to name them unambiguously, with clear context (position/hierarchy in the model) and unique name.

3.2 Tool chain capabilities to support requirement management

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 chapters through main scenarios.

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).
  - **OpenETCS example**: visualize one openETCS software requirement, its statement and upstream traces up to Subset 026.
- 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)
  - **OpenETCS example**: list OpenETCS SW 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, from risks and hazards analysis (safety requirement) or from software design. 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 engineering levels (scopes) that can be defined by end users.
  - **OpenETCS example**: allocation of a newly created requirement to the OpenETCS system level, Onboard Unit SW level, OB SW component level or OBU SW component interface level .
- 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 of a given scope (level) 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 (system input activities) or two versions of OBU SW 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 (System specification activities) or ReqIF format (OBU SW requirements). 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.

3.3 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 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 openETCS SW functional 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 openETCS SW 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 links 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 analyzed by project manager and/or quality team

4 Short term development priorities and associated practical scenarios concerning traceability

OpenETCS activity targets long term objectives (recalled in introduction) but has also to fulfill short term target with demonstrators for ITEA2 project within limited remaining efforts and time. This chapter explains the priorities put on development (design and verification activities) to reach this short term target and provides practical scenarios with focus on traceability.

4.1 Main design priorities and current requirement organization

From WP3 point of view, most important development chain concerns ETCS OBU formal model and its links to the 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 SRS Subset 26 requirements, then it will be possible to establish that software code generated from that model also satisfies those requirements.

Full process recommended by CENLEC engineering standards implies deriving all stakeholder reference requirements (including SRS subset 26 , openETCS API and user stories) into OpenETCS requirements layered in system, subsystem and software levels. See figure 7 for ideal requirement management organization.

But most openETCS requirements starting from ETCS system down to software level are already present in SRS subset 26 through the different chapters. So requirement analysis and derivation process mainly leads to propagating existing requirements from SRS subset 26 document to OpenETCS requirements and the value of such transformations is low compared to the efforts for propagation and allocation to levels. So idea is to use SRS subset 26 as openETCS requirements and use the chapters structure to differentiate between levels. They are translated into .reqIF format so that requirements can be managed by requirement management tools. Other reference requirements are imported in the openETCS reference configuration (baseline) as documents.

Figure 8 illustrates current organization for requirement management.

SysML model, used to capture and describe architecture from System level down to software internals, is considered as an intermediate means in the sense that it helps producing the OBU scade formal model but might be not maintained once OBU model is finalized and can be traced directly to the SRS Subset 26.

But currently, SysML model plays a role in this definition and so we need to create traceability links with it so that we can verify that architecture conforms to the SRS subset 26.

Figure 9 illustrates all required traceability links needed to achieve current design verification and highlights those conisdered with highest priority (arrows with largest size) for short term.

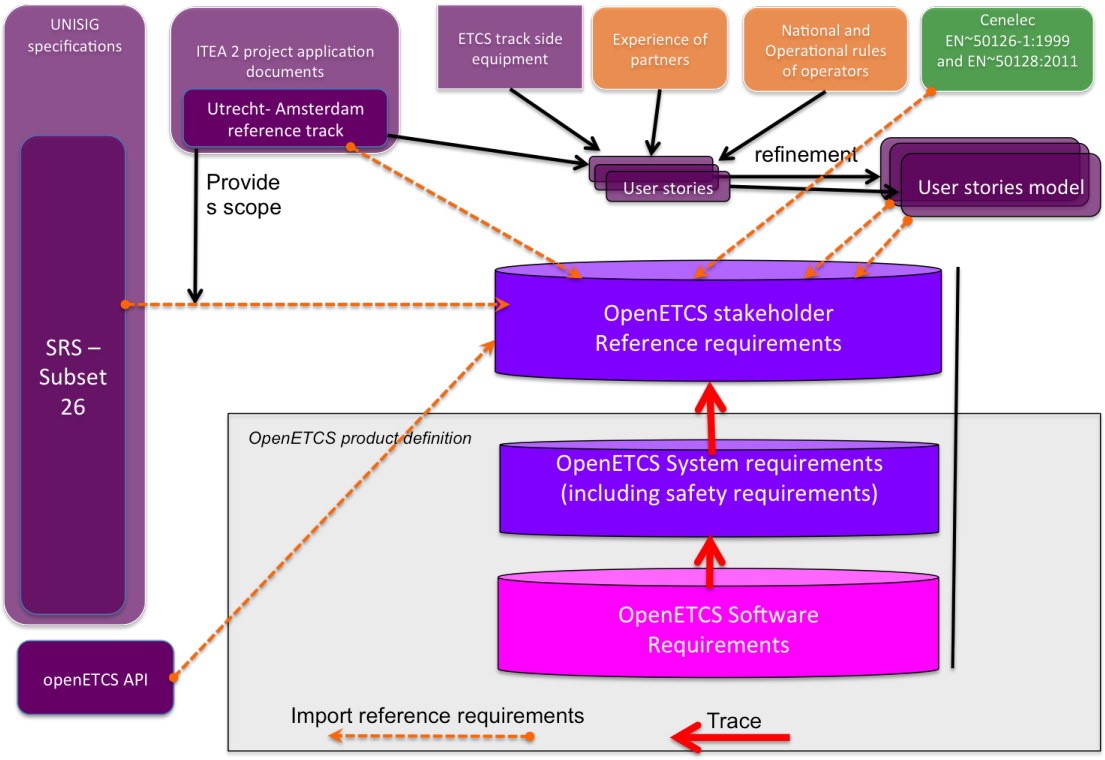


Figure 7. OpenETCS ideal requirement management organization

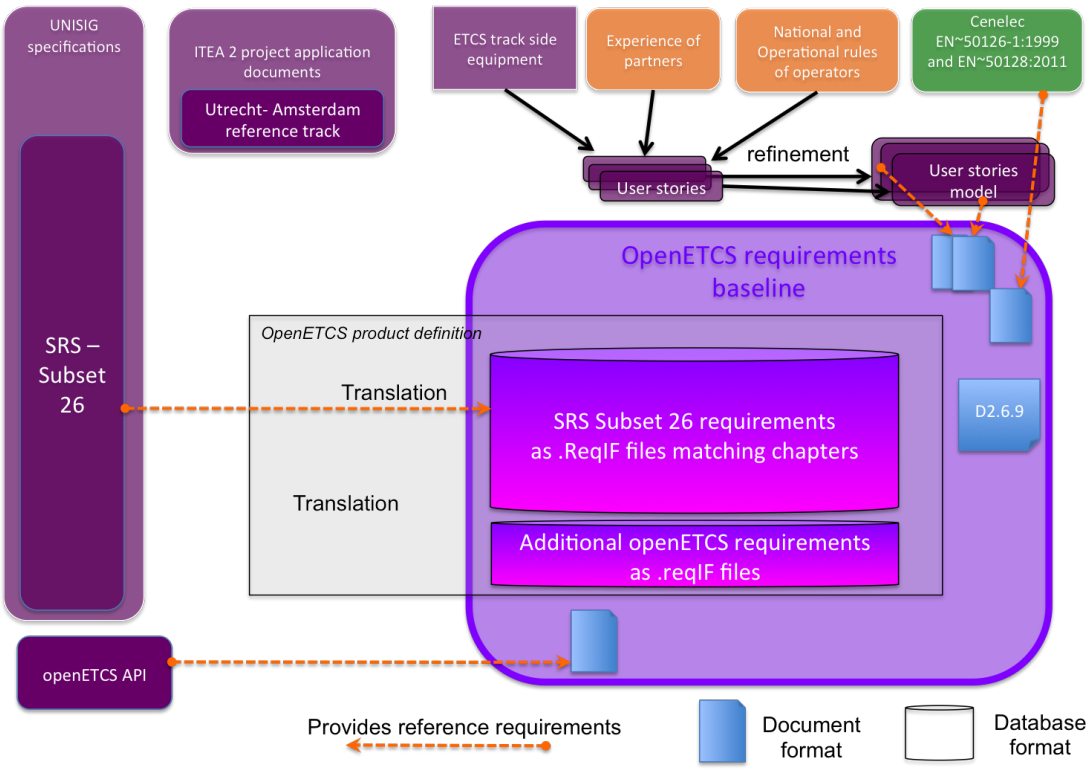


Figure 8. OpenETCS current requirement management organization

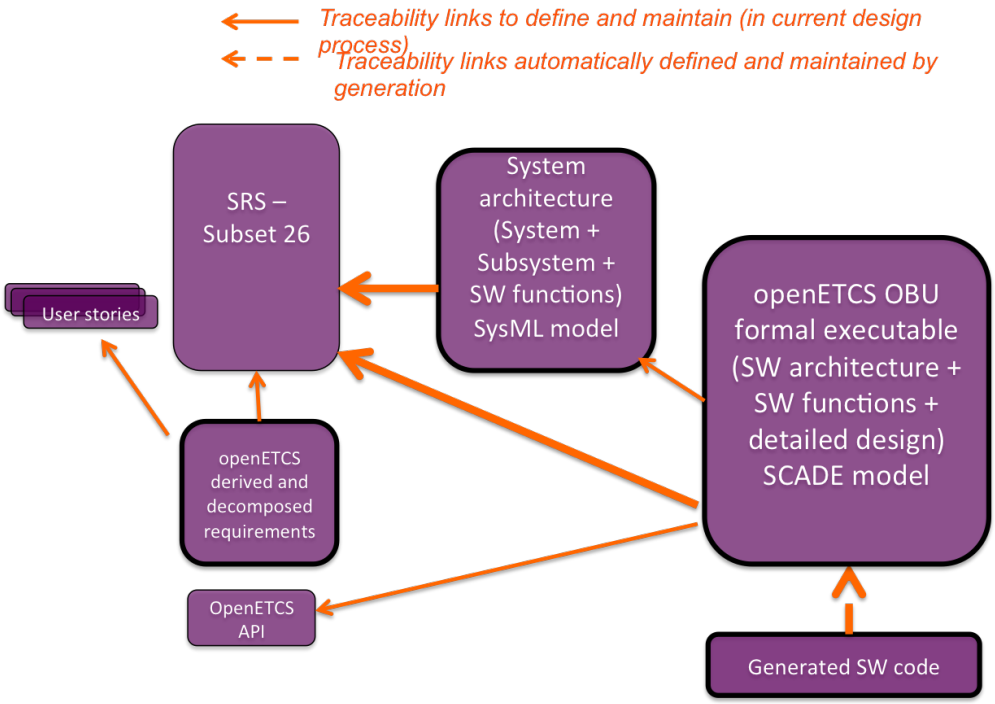


Figure 9. OpenETCS traceability chains for current design with highlight on main priorities

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.

4.2 OpenETCS traceability main scenarios

Traceability main scenarios are grouped by phase according to the openETCS software development process introduced in previous chapter.

We recall illustration of software development process below to ease reading of next paragraphs.

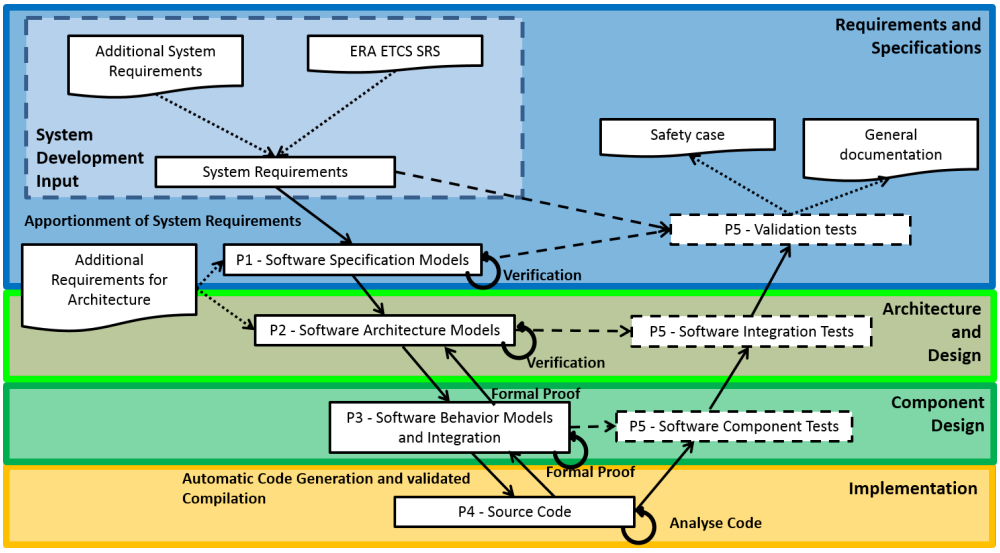


Figure 10. OpenETCS software development process

4.2.1 P1 Software specification Phase

[6] recalls P1 scope and current support for verification. *The phase P1 Software Requirement Phase covers the development activities required by the EN 50128 during its lifecycle phase Software Requirements Phase (7.2). The SUBSET 26 and further System Specifications are the Input of this work. The resulting ReqIF Requirement files shall represent the Software Requirement Specification. Software Overall Test Specification will be derived with respect to these requirements, but a method is not defined at this point.*

Currently .ReqIF Requirement files are translated from SUBSET 26 and there is no derivation or special tag to identify SW requirements. So software overall test specification shall first include this derivation of software requirements in order to conduct test overall SW test specification.

Note: SysML architecture model provides a good input for such derivation to SW requirements by identifying system, subsystem and SW engineering levels through SysML Block Definition Diagrams and Internal Blocks Diagrams. But currently this model does not lead to the creation of software requirements in the database.

Figure 11 illustrates what could be done.

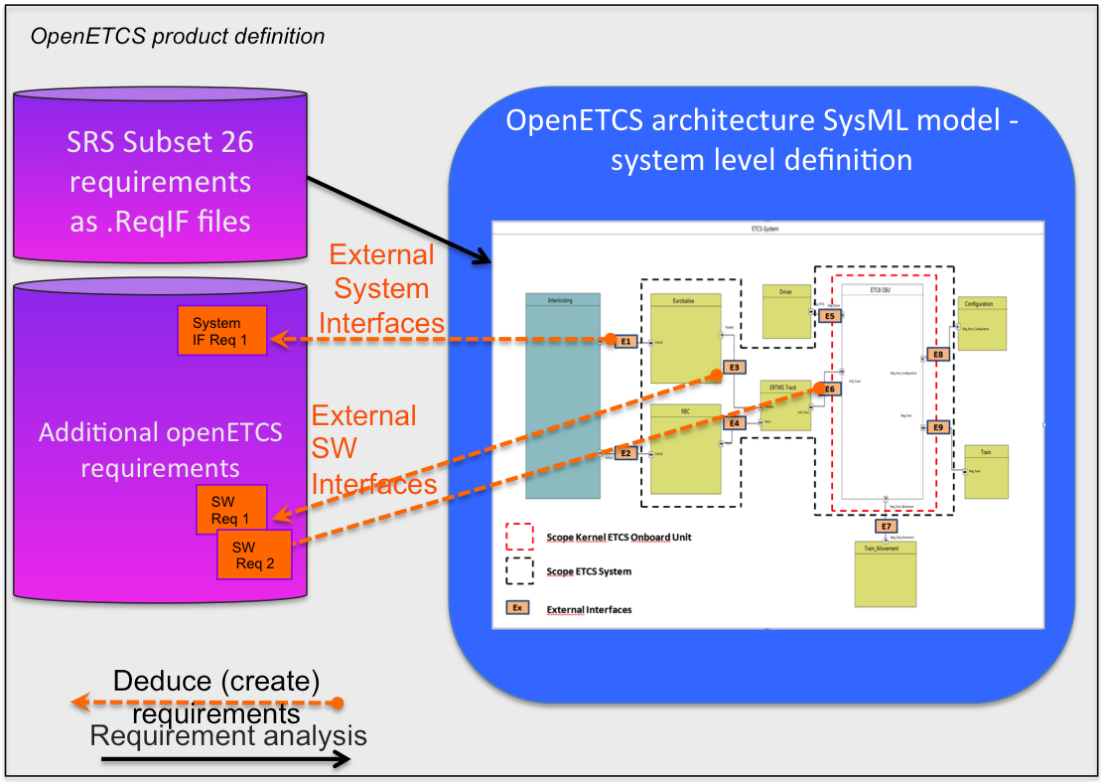


Figure 11. Requirement derivation process through SysML architecture model

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

Note: concerning Hazard and Risks analysis activity, it has been performed only on a subset of project scope and through a Proof of concept (see 2.4) that has demonstrated principles and practical feasibility with methods and tools. But this PoC has not yet been extended to the whole scope of project. For short term development, systematic identification of safety requirements

and derivation into software requirements are not the main priority now that principles have been demonstrated with success.

Once SW requirements are clearly identified (either created by derivation or tagged in .reqIF files or in other existing documents), it becomes possible to define associated verification means, among test specification, inspection, analysis or certification and associate verification specifications. Those verification specifications shall mention the openETCS concerned SW requirements so that we can check relevance and accuracy.

In terms of traceability it means that we shall be able to show bi directional traceability between openETCS SW requirements and associated verification specifications. Usual way is to deliver a matrix for that. Then, reviews can be done with matrix as input to ensure that verification specifications really verify mentioned requirements and that requirements are fully verified by associated verification specifications.

4.2.2 P2 - Software Architecture Modeling Phase

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

It shall take software requirements as inputs.

Note: as mentioned in P1, currently .ReqIF Requirement files are translated from SUBSET 26 and there is no derivation or special tag to identify SW requirements. So there is no clear SW requirements reference against which we can rely to design and verify architecture.

SysML architecture model initiated in P1 is completed to represent Software Architecture Specification down to SW components and their interfaces. It does not take software requirements as input but takes upper level architecture: that approach at least ensures consistency between engineering levels. Then this sysML model is reflected (translated) into a SCADE integration model that will be completed and refined during P3 phase.

Concerning architecture verification, nominal method would be to verify SysML architecture model against SW requirements but they cannot be clearly identified so currently verification is performed directly against SUBSET 26.

Concerning results of architecture, SysML architecture model provides a good input for creation of first SW component and SW component interface requirements through SysML Block Definition Diagrams and Internal Blocks Diagrams. But currently this model does not lead to the creation of requirements in the reference requirements database.

Figure 12 illustrates what could be done.

4.2.3 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 overall SCADE model initiated during P2 is refined and enhanced to completely represent Design and Interface Specifications as well as all Component Design Specifications. Source Code in C is generated

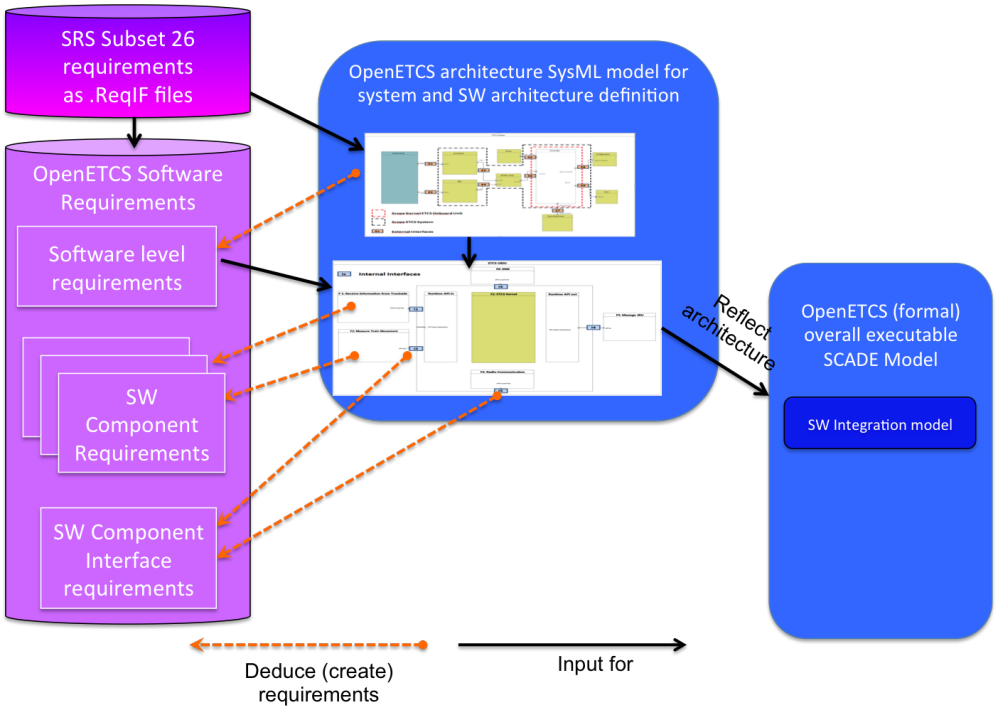


Figure 12. Software requirements initial definition from SysML model

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.

As this overall SCADE model is composed of several components described by lower level SCADE models and their interfaces, it specifies integration to be performed it can be used as input for SW integration (see P5). This is a "meet in the middle" approach taking as inputs SysML architecture model (top down approach) and functional SW component detailed models (bottom up approach).

Note: **this Scade overall SW integration model does not appear on current Software development process figure 10 in Architecture an Design box and should be present there because it is an important artifact used both as integration of behavioral models (rising part of V cycle applied on models) and as input to prepare integration of software**. Said it differently, there is a small "V" starting with P2 in Architecture and design, descending with P3 in Component Design and rising again in Architecture and Design with P3, still in the descending part of the global V life cycle.

SCADE integration model describes structural but also dynamic integration of component models. It includes at least following elements to consider:

- conditions of activation and deactivation of functions (described by component models) through use of modes,
- Scenarios of functions with control flow including sequencing, branching, merge, fork and synchronization,
- Arrangement of time events (scheduling), management of asynchronous events
- Error/exception management to ensure reliability

- Technical conversion functions to manage exchange of data available in different types/formats
- Conformance with SysML architecture model?

All those SCADE models (low level behavioral models and integration models) provide good input for creation of additional SW component and SW component interface requirements from SCADE operators, blocks, automatons and data types.

The software architecture and design document produced from those SCADE models is also another good input for creation of requirements.

But currently neither SCADE models nor Architecture and Design document lead to the creation of requirements in the reference requirements database.

Figure 13 illustrates what could be done in the future to get a clear SW definition.

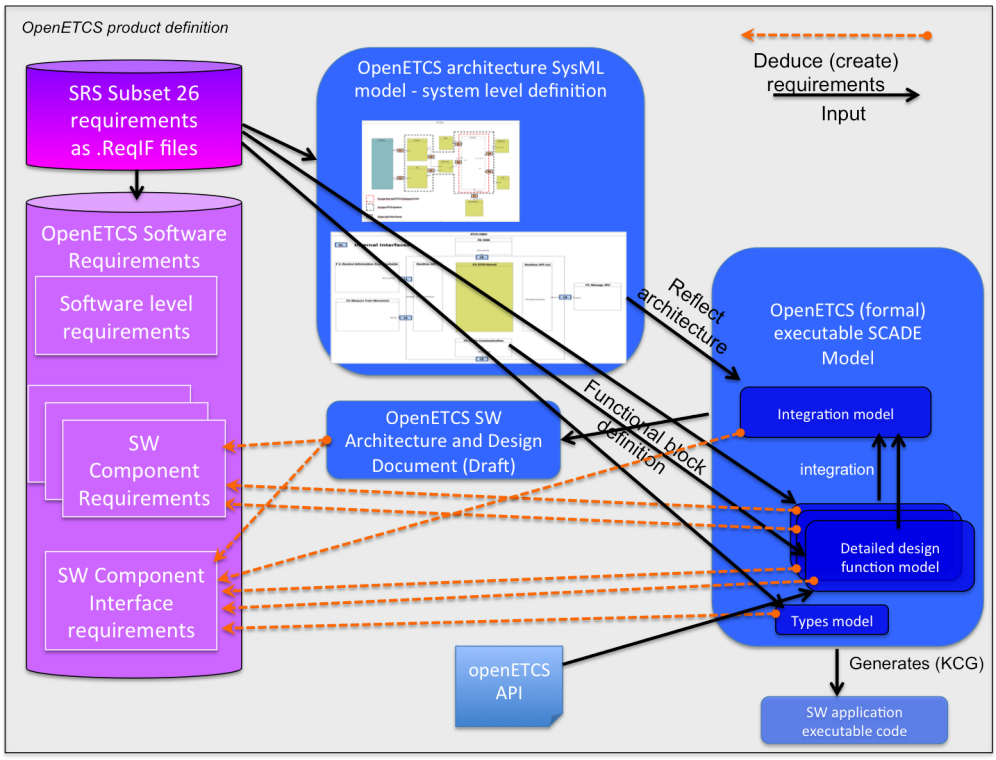


Figure 13. Software requirements identification from SCADE models and from Software Architecture and design document

4.2.4 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.

Currently there is no precise list of traceability links to define and maintain in the frame of this phase.

5 Tool chain current physical architecture solution to support traceability

5.1 Overview of current traceability architecture solution

This current solution consists in handling requirement management and traceability links through different tools and technologies. The strategy of this approach consists in maximizing efficiency of the different concurrent activities through using tooling locally integrated in the different environments used in the project over its development life cycle (the 5 phases).

As a summary view (will be detailed later) we find: - P1: ProR is used to support requirement management around a .ReqIf files database - P2: Scade System is used to edit SysML architecture model from system environment down to software definition and its external interfaces. This model can be displayed in Papayrus SysML and then used by PolarSys ReqCycle tool to create links with .reqIF requirements. - P3: . ReqIF reference requirements are imported in SCADE and can be traced through the ANSYS RM Gateway based on Reqtify solution. - P4: SCADE KCG is used to support generation of C source from SCADE models. - P5: other tools (to be completed)

Figure 14 illustrates current traceability architecture with associated set of tools, scripts and procedures.

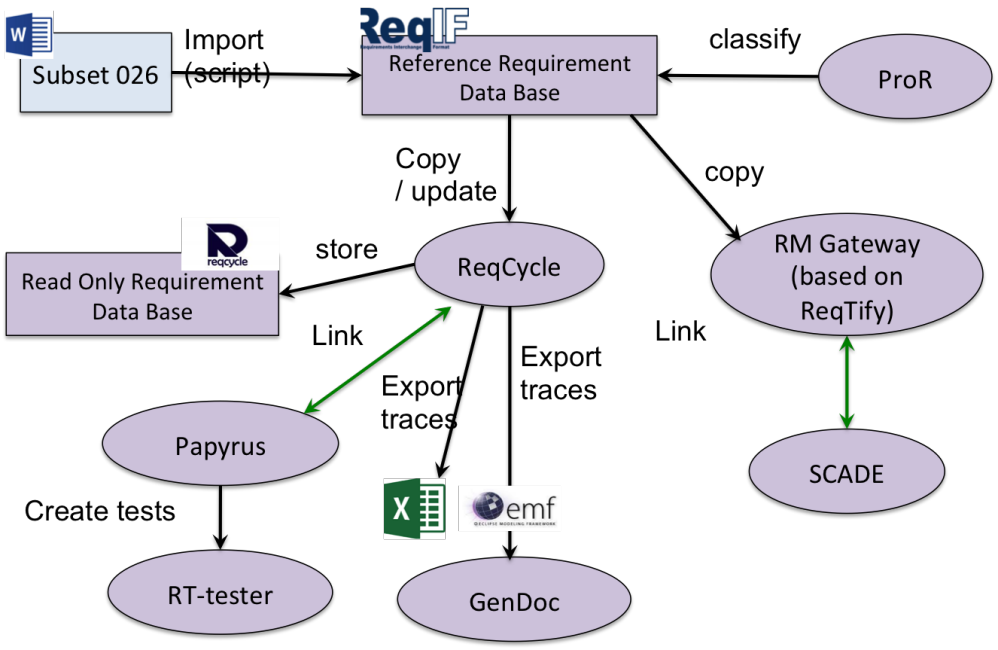


Figure 14. Traceability architecture first solution

Note: when requirement management tools complete imported requirements, 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.

5.2 P1 phase: ProR, Scade System, Papyrus and ReqCycle

5.2.1 Import of reference requirements through a Script

Reference Subset 26 input document is converted to a set of ReqIF format files and informally analyses. The analysis specifies relationships between requirements and revised parts of the requirements to obtain a detailed and atomic abstraction level matching the different requirement engineering levels required by Cenelec standards.

A script is in charge of converting input SRS subset 26 document to requirements in .ReqIF format files. Each requirement has an unique identifier. These identifiers are fixed and cannot be modified by any other tools. See for detailed description of this script, challenges to solve and solutions.

5.2.2 Classification of reference requirements through ProR

ProR is the tool in charge of supporting the requirement classification from this initial set of SUBSET 26 requirements:

- filter of requirements to take into account for openETCS project baseline (not all SUBSET 26 requirements are part of openETCS baseline)
- tag requirements that are considered as software requirements (normally there should be a derivation process that would create SW requirements but it is not the case currently and SW design takes as inputs SUBSET 26 requirements)

Currently there is no classification of reference requirements and .reqIf files are an exact translation of the document.

5.2.3 Creation of additional requirements through ProR

ProR is also in charge of supporting creation of additional requirements coming from:

- hazard and risks analysis (safety requirements) but we saw that the safety process has been validated through a proof of concept but not yet deployed for the whole project.
- decomposition (improvement of requirement quality) but this is not done directly in .reqIF files. There is a process called RFC (Request For Comment) in charge of collecting deviations to the SRS subset 26 detected during design.

So finally .reqIF files only contain SRS subset 26 requirements and no additional requirements are created for now.

5.2.4 Link requirements and overall test specification with ReqCycle

As mentioned in previous chapter, overall test specification is not yet defined but should be linked to the SW requirements and each SW requirement shall precise its verification means (including test specification) and be linked to verification specification (test specification if it is the prefered verification means).

It first means that all requirements shall have an attribute "verificationMeans" taking as possible values "test", "inspection", "analysis" and "demonstration" according to verification plan. We suggest using ProR tool to add this attribute.

It also means that each ReqIF requirement shall have links with at least one of the planned verification means. Tool chain shall support creation of those links and export as a traceability matrix. There are mainly two options with open source solutions:

- either use ProR to manage the links with verification specification (mainly test specification) and export associated traceability matrix.
- else use PolarSys ReqCycle tool to manage those links and export traceability matrix.

A few words about PolarSys ReqCycle: this tool was not mature at the beginning of the openETCS projet and was not really available as official open source component from Eclipse. Now this is an official solution from PolarSys industrial Working group from Eclipse foundation and it has followed official Eclipse development and release processes that ensure minimum of quality and transparency in addition to IP analysis. Official description is available here: https://www.polarsys.org/projects/polarsys.reqcycle

In addition, 0.8.0 release has been largely tested and is considered as stable. There is Wiki documentation (with development process, user and some tool requirements, installation and tutorial) available here: https://polarsys.org/wiki/ReqCycle

ReqCycle can export traceability links to .csv file and to EMF format model, that can then be used as input by Eclipse gendoc tool to produce an industrial quality level documentation.

Prefered solution ProR is the simplest approach to get .ReqIF requirements and it can create links, but this approach requires some additional plugin to analyse test definitions from verification specification document.

ReqCycle has to import .ReqIF requirements first before creating traceability links but it is not an issue because there is a ReqIF import connector. Concerning tests, it has a generic document connector able to analyse and extract test definitions if they can be identified through regular expressions, document styles or combine both technics. This connector can also use OCL language and associated queries to identify tests from spreadsheets.

Finally we suggest using PolarSys ReqCycle to support management of links between requirement and test specification because it has all native features/capabilities to import requirements, import test definitions, create links and export them in a .csv matrix.

5.2.5 Update of SRS subset26

P1 is also concerned by potential changes in reference requirements coming from a new version of SRS subset 26. Any new version should be analyzed for potential impacts on OBU Software. So the tool chain shall be ready to support such update.

Practically it means that it should be possible to compare two versions of .reqIF files created by same import script applied on two different SUBSET 26 versions. That comparison shall lead to a list of deleted requirements, modified requirements (changed text) and new requirements.

From this analysis, the tool chain shall help providing the list of all OBU SW requirements concerned by deleted or updated requirements in order to check associated design and verification specifications (including tests). It means reviewing all traceability links associated to modified .reqIF requirements.

ProR can compare two .reqIF files and then identify changed and deleted requirements. But in order to analyse impact on design and overall test specification, ProR shall access to all those traceability links associated to reference requirements, what is not the case currently.

Can ReqCycle help for that? ReqCycle can manage updates of its different requirement sources and list impacts on traceability links. So it might work but under the condition that all tracebility links are captured (created or analysed) by ReqCycle. That is not the case currently.

So this point remains open.

5.2.6 technical challenges

5.2.6.1 Script description

The *Subset26Import* script generates a hierarchical tree of all traceworthy artifacts in each chapter of subset-026. Each artifact shall be uniquely addressable via a tracestring.

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 15. 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]`

5.3 P2 phase: ProR, Scade System, Papyrus and ReqCycle

Concerning P2, main traceability links concern verification of SW architecture.

5.3.1 verification of SW architecture through ReqCycle

Architecture model is defined with a subset of SysML language (blocks definition and internals) through Scade System tool. We want to trace this architecture to the reference requirements that are imported (translated) subset 26 requirements.

3 options could be considered to support creation of this traceability:

1. **traceability in Scade System environment through the Ansys RM Gateway based on Dassault Systems Reqtify product**: .ReqIF requirements are imported in the RM Gateway (through Reqtify configuration) and Scade System provides a way to reference those requirements from blocks or connectors that define architecture. The RM Gateway creates the links and exports a traceability matrix.

2. **traceability in ProR through dedicated ProR-Papyrus proxy module**: SysML model created with Scade System can be displayed with Papyrus open source tool. ProR is used to create traceability links to model elements from Papyrus thanks to a special configuration and a proxy module (see https://github.com/openETCS/toolchain/wiki/User-Documentation) .

3. **traceability through PolarSys (Eclipse-based) ReqCycle tool**: we use same previous approach to see SysML model initiated by Scade System in Papyrus tool. Then .ReqIF files are imported through a ReqCycle import connector and made available as ReqCycle requirements (in pivotal eclipse modeling format). Traceability links are created by traceability creator or Drag and drop features available in ReqCycle to link reqCycle requirements and any EMF model including Papyrus SysML. Requirements and traceability progress can be visualized in ReqCycle and traceability data can be exported to .csv file.

Option 1 relies on closed source technology (RM Gateway) and there are licensing issues. Option 2 relies on specific proxy ProR module developed during the project: require adapters from both ProR and Papyrus and maintenance is not guaranteed over time (not core part of both roadmaps). Option 3 relies on native features from ReqCycle tool: a ReqIF connector to import .ReqIF requirements and native commands like traceabilityCreator view to trace requirements with EMF models including Papyrus (see 5.3.2)

Option 3 is considered today as best option to support traceability concerning SW architecture verification.

Architecture is initialized by SysML model and then reflected and completed in SCADE model. So question becomes: what is input for traceability with requirements? SysML model or SCADE model?

So we suggest creating links from SysML architecture model. With that approach we can use ReqCycle (tool has been briefly presented previously - see 5.2.4).

5.3.2 Creation of traceability links between a requirement and a Papyrus SysML with ReqCycle

Figure 16 illustrates a simple way to create a link between a reference requirement and a model element through ReqCycle traceability creator view.

5.4 P3 phase: ProR, Scade suite and ???

SysML brings top level decomposition architecture staring from openETCS system while SCADE model brings bottom up SW architecture coming from integration of behavioral SCADE models. So when we look at SW detailed architecture and at behavioral models, the reference is SCADE, not SysML. So we need to be able to trace all detailed design elements from SCADE behavioral models and all other design elements associated to their integration at upper level. All those elements have to be traced to the reference requirements.

From SCADE the only serious option today is to use the proprietary ANSYS RM Gateway but there is licensing issue for some partners. So currently this is not ideal solution and there is no real support of traceability for SCADE model elements today.

One option exists but it has not been investigated seriously for now: using SCADE Eclipse API to export SCADE models to Eclipse EMF Format and then use ReqCycle to perform traceability

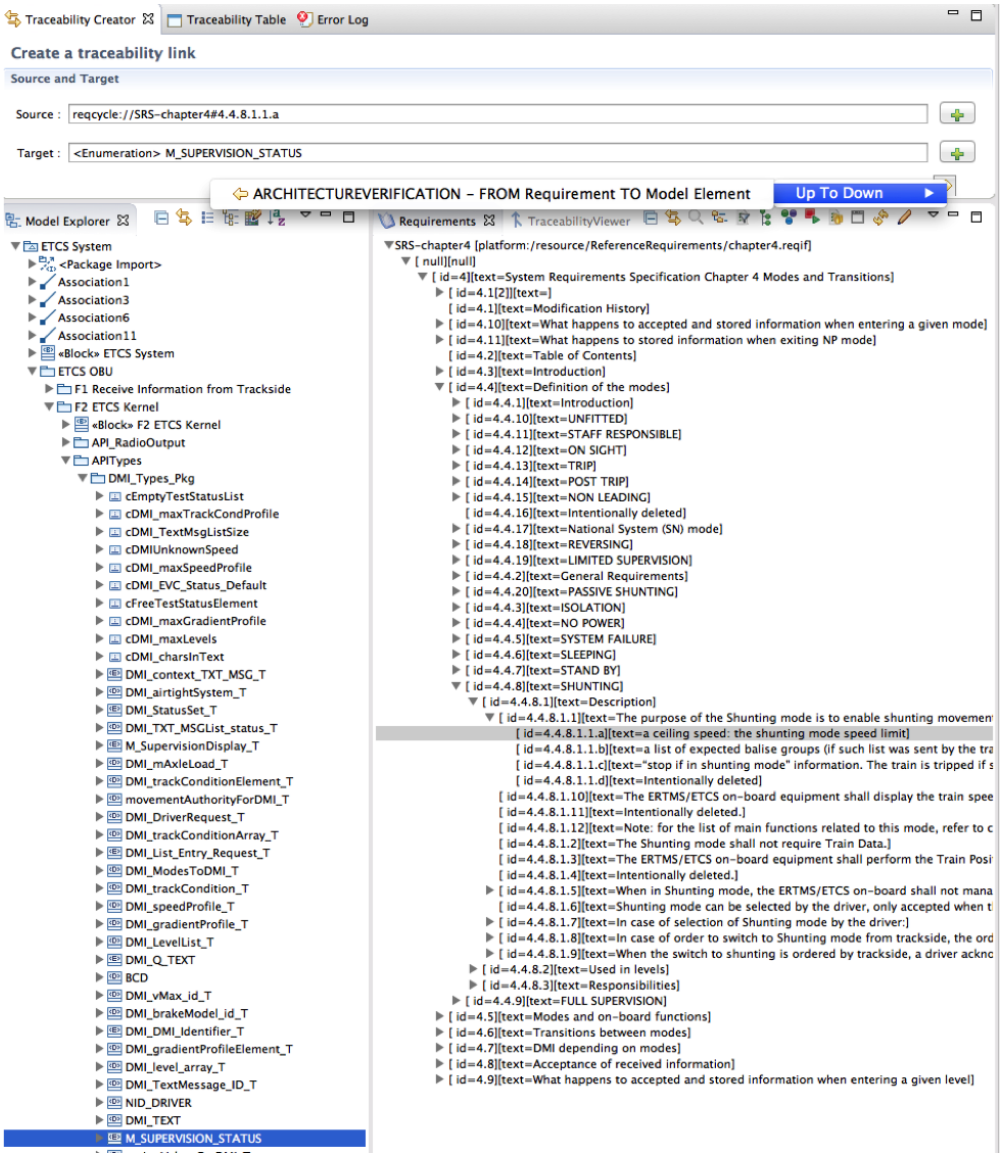


Figure 16. Creation of links with Reqcycle between requirements and Papyrus

as ReqCycle has native commands to create links between any requirement and EMF model elements. We do not consider this option in this document and rather select it as an improvement axis.

5.5 P4 phase: SCADE suite and Scade KCG

Traceability between SCADE model and C source code is ensured by SCADE KCG.

5.6 P5 phase: Verification and validation of SW product

Currently no traceability relations have been clearly defined.

6 Limits of current traceability architecture and axes of investigation for improvement

6.1 Current identified limits

Before looking at possible alternative software solutions to support traceability in the different phases, let us come back to the choices in terms of artifacts to trace.

6.1.1 Limits concerning the baseline of requirements used as input for traceability

Currently only SRS subset requirements are considered in the creation of traceability links. But there are also other sources of requirements including openETCS API, CENELEC standards and user stories. How to ensure traceability to those artifacts if they are not integrated into the reference requirements database?

Secondly, with *traditional* SRS requirements derivation down to SW requirements, the method consists in decomposing functional requirements and then allocating them to lower level design elements and so on down to Software level. At Software level, then we trace software architecture elements to software requirements through a "satisfy" semantics trace. Finally we have different steps in the creation of traceability links and all created links have "simple" semantics ("satisfy" or "allocation") that can be validated quite easily.

in P1, we trace directly architecture elements (SysML model) to SUBSET 26 requirements instead of tracing them to openETCS SW requirements derived from System requirements also derived from SUBSET 26 requirements. The consequence is that the trace links are harder to validate because they sometimes connect elements of very different engineering levels.

Finally we strongly recommend three things: - think about other scripts or solutions to import other reference stakeholder requirements including at least openETCS API and user stories (CENELEC standards can be traced manually) to improve traceability to its reference sources. - add a tag to all reference requirements with identification of a level to clearly identify SW requirements and those that are from a higher level. - provide a rationale to the trace link, at least concerning all requirements that are not clearly identified as software requirements, to give understanding elements to the future reviewers (next review or perhaps in several months when Subset 26 version will change).

6.1.2 Limits concerning additional requirements

We already mentioned the SRS requirements derivation process not done but there are also other requirements that should be created during SRS analysis: safety requirements, as a result from risk and hazard analysis.

When considering that openETCS are only requirements coming as inputs, we can take the assumption that we never change them and consider all requirements as "read only" requirements. We do not look into the issues of managing (import, export, synchronization...) additional requirements potentially created by other tools like EventB or another tool.

6.1.3 Limits concerning traceability with SysML elements

For verification of SW architecture we have considered that SysML model was representative and that we could then create traceability links with sysML model elements. What happens if overall SCADE model is not aligned with SysML model concerning architecture? then traceability links with SysML model will be false and review cannot see that.

So, in order to continue trusting links done with SysML model we must ensure that architecture designed with SysML is reflected in overall SCADE Model and that any modification on architecture done at SCADE side is reflected back to the SysML model. Said it differently there must be architecture synchronization between both models. If there is no way to check that synchronization, some links done to verify architecture might be wrong without ability to notice that.

6.1.4 Limits concerning traceability with SCADE elements

Currently there is no satisfying solution to trace SCADE model elements to reference .ReqIF elements. Some use the RM gateway but others do not for licensing reason and finally it puts some risk on the project not to be able to verify software detailed design.

6.1.5 Limits concerning management of all traceability links

We saw in previous chapter that in case of update of SRS Subset 26 ProR will be able to provide a diff between requirements but will probably not be able to list all associated traceability links that have to be checked and handled (removed or confirmed or completed). That is another issue to investigate.

6.2 Investigation axes for improvements

We suggest different investigation axes to manage main limits identified previously. Those investigations will be detailed and will lead to results reported into an other document.

6.2.1 Ability to export Scade model as Eclipse EMF model

Esterel Technologies explains in their SCADE documentation that it is possible to export SCADE into Eclipse Modeling Framework format through their Eclipse API: see http://www.esterel-technologies.com/products/scade-suite/software-prototyping-desgin/scade-suite-advanced-modeler/

With that approach it would become possible to see SCADE exported models in the Eclipse environment and create links through ReqCycle traceability engine (ReqCycle can trace requirements to EMF models without needing a new adapter).

This is the most urgent investigation axis to look at. If it provides good results, then ReqCycle can be used to manage most important links in openETCS project and can be considered as reference tooling for traceability with ability to aggregate all links. It simplifies architecture and answers to the last two limits identified previously.

6.2.2 Combined solution for requirement management and traceability tools

Concerning requirement management and traceability there exist mainly two open source Eclipse based solutions: ProR and ReqCycle. RMF/ProR is the natural choice for OpenETCS to support

requirement management while ReqCycle is a good choice to support management of traceability links (creation, capture, visualization and export).

It would be interesting to investigate on a second architecture solution combining ProR and ReqCycle as reference tooling to manage both requirements and traceability links in an integrated way.

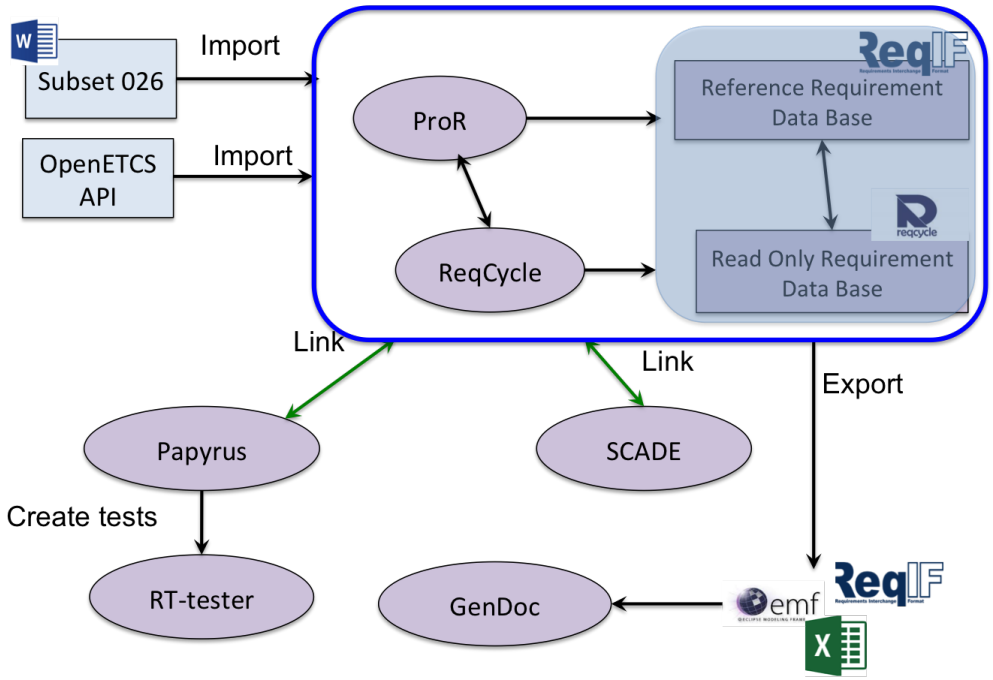


Figure 17. Traceability architecture second solution

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 **??**) 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.

6.2.3 ReqCycle used as centralized solution and ProR for requirement edition

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). ProR is used to classify and edit requirements instead of ReqCycle basic requirement editor.

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 that delegates requirement edition to ProR.

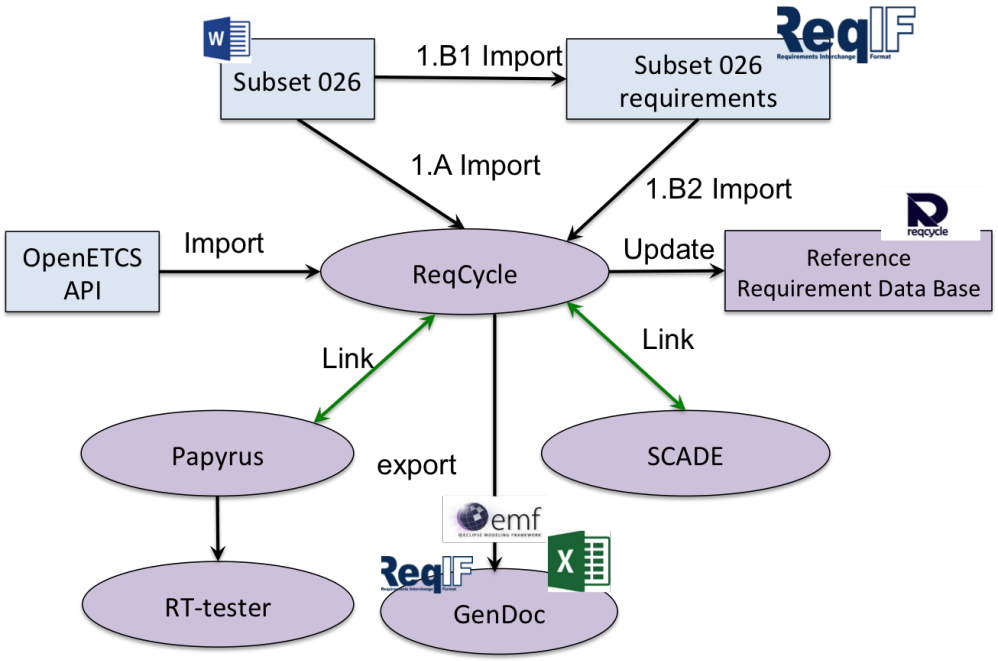


Figure 18. Traceability architecture third solution with ReqCycle only

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] openETCS WP2. *OpenETCS higher level requirements*, 2015. https://github.com/openETCS/requirements/blob/master/D2.6-9/D2_6-9.pdf.

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

[4] 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.

[5] 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.

[6] openETCS WP4. *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.