

Work-Package 3a: “Toolchain”

## openETCS Toolchain WP Description of Work

Michael Jastram, Christophe Gaston, Marielle Petit-Doche, Jan Peleska, Jonas Helming and WP3a participants

November 2012





# openETCS Toolchain WP Description of Work

Michael Jastram

WP3a Leader

Christophe Gaston

WP3a.1 Task Leader (Core Tool Chain Analyses and Recommendations)

Marielle Petit-Doche

WP3a.2 Task Leader (Supporting Tools Analyses and Recommendations)

Jan Peleska

WP3a.3 Task Leader (Define and Develop Tool Chain)

Jonas Helming

WP3a.4 Task Leader (Develop Open Source Ecosystem)

WP3a participants

OpenETCS

## Description of work

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.



**Abstract:** This document contains the description of work planned for WP3a. This revision is necessary, as the workpackage from original ITEA proposal was split. This document will be the foundation for a revision of the ITEA proposal.

The revised DoW consists of four chapters for the four Tasks of WP3a.

**Disclaimer:** This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 – (cc by-sa 3.0)

THE WORK IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS 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/>

# Table of Contents

Introduction .....	4
1 Core Tool Chain Analyses and Recommendations.....	4
1.1 Identify and define the potential modelling languages .....	5
1.2 Identify and compare existing modelling tools .....	5
1.3 Identify the tool platform .....	6
2 Supporting Tools Analyses and Recommendations.....	6
2.1 Identify and define the supporting tools and methods .....	7
2.2 Model Transformation and Code Generation .....	7
2.3 Requirement traceability.....	8
2.4 Validation and Verification Tools .....	8
2.5 Safety analyses Tools .....	10
3 Define and Develop Tool Chain .....	10
3.1 Overall Tool Architecture .....	10
3.2 Development Infrastructure .....	10
3.3 Decomposition and Distribution of work .....	10
3.4 Inputs and Outputs for Section 3 .....	11
4 Develop Open Source Ecosystem.....	11

## Introduction

This Work Package will provide the tool chain, based on formal methods, that is necessary to design, develop, verify and validate the ETCS system. This tool chain will encompass all description levels of the system design from holistic viewpoint to code generation. Specific attention shall be paid to the semantics and the traceability of each part of the chain. The formal specification will be used further for verification and code generation. Other uses of the formal specification are conceivable.

The tool chain must at least support the following tasks:

1. Support the writing of the formal ETCS system description, which may include a wide range of artefacts, from requirements to actual code (and everything in-between). To achieve this, it will include elements like means of description (e.g. modelling languages), graphical or textual editors, version management, etc.
2. Support code generation.
3. Support actions like execution, debugging and simulation of some or all elements of the system description.
4. Support the generation of elements that are needed for testing, like test cases, test data, test oracle, test procedure execution, etc.
5. Support the verification and validation of the various artefacts, including the formal ETCS specification against the textual ETCS specification.
6. Support tracing of artefacts across all tools and build steps, as needed.
7. Analysis of the system description, like transitions and cooperations between models.
8. Integrate the tools as needed to support the users as efficient as possible.

The tool chain definition will benefit from other R&D projects and off-the-shelves tools. The semantics of the modelling languages shall be carefully studied.

The first goal of this WP is to identify sets of consistent languages and tools enabling the design of the system. This will be done in close collaboration with WP2.

In order to be able to progress without depending too much on WP2 requirements and other deliverables, the subtasks shall make use of prototyping in order to gain knowledge regarding the possible modelling languages and tool platforms.

Compliance with CENELEC standards EN 50126, EN 50128 and EN 50129 could have a significant impact in terms of workload. Moreover it should be included at the beginning of the tools development in the Development Plan or in a Safety or Quality Plan in close collaboration with WP4.

## 1 Core Tool Chain Analyses and Recommendations

The first task is the core tool chain analyses and recommendations. It is concerned with the languages themselves, the tools for authoring, as well as the methods used for formalising the requirements and specifications.

The core tools will be complemented by supporting tools, as outlined in Section 2. Specific attention shall be paid to the semantics and the traceability of each part of the chain.

### 1.1 Identify and define the potential means of description

The goal of this task is to identify the modelling languages that could fit the requirements associated to the specific needs of ETCS design and railway norms. Those needs will be defined by tasks of WP2 (in particular D2.3.1 - Set of requirements on modeling and D2.2.1 - Process definition). Depending on recommendations of WP2, several languages may be necessary to handle the different levels of abstraction of the whole design process. For each candidate, a small subset of the ERTMS specification will be modelled. The languages may have to be adapted in the process. The identification and definition will distinguish between wide-spectrum modelling languages suitable for a wide variety of modelling domains such as UML, SysML, B, and domain-specific languages (DSL) designed and optimised for application in a specific application domain only. For wide-spectrum languages their metamodels<sup>1</sup> will be analysed with respect to their expressive power and resulting adequateness for designing ERTMS models. For DSL candidates the associated meta-metamodels<sup>2</sup> will be analysed with respect to their capabilities to support language extensions that may become necessary for novel releases of the ERTMS specification in the future. Since no language is universal (i.e. able to address all aspects of design needs) the proposed approach is likely to involve several modelling languages supporting different viewpoints and working at different levels of abstractions. With this kind of approach, we will need to check the compatibility of the semantics of the modelling languages that address overlapping viewpoints. There are two problems here. First, when dealing with an heterogeneous specification, we need a common semantical basis to check the compatibility of the models. More pragmatically, when we deal with two models (expressed in a different language) that describe the same part of the system, we need to show that they are consistent with each other. Candidate languages will be subsequently evaluated against the requirements from WP2. If a suitable language is identified, but no partner steps up to model the prototype, it will not be considered.

Input:	WP2: List of suitable languages (based on State of the Art Analysis)	Nov-12
Input:	WP2: Small subset of ERTMS requirements that is representative	Nov-12
Input:	Those WP2 Requirements that are sufficient to evaluate a target language	Jan-13
Output:	Formal Model representing the sample spec, one for each candidate	Mar-13
Output:	Documentation of the changes to each language used (if any)	Mar-13
Output:	Evaluation of the models against the WP2 requirements	Apr-13
Output:	Decision on the final language choice(s)	May-13

### 1.2 Identify and compare existing modelling tools

Corresponding to Section 1.1, the objective of this subtask is the identification of the core modelling tools (analysis and other supporting tools will be the subject of Section 2), based on the analysis from WP2, by using it for the prototyping described in Section 1.1.

<sup>1</sup>Recall that metamodels specify the syntax and static semantics of a modelling formalism.

<sup>2</sup>Recall that the meta-metamodel specifies the capabilities to define language elements and their static semantics in a DSL.

The experience with the tools will be recorded, and the tools will be evaluated against the requirements from WP2. In particular, the compliance of the candidate tools with respect to EN50128 constraints will be investigated.

Input:	WP2: List of suitable tools (based on State of the Art Analysis)	Nov-12
Input:	Those WP2 Requirements that are sufficient to evaluate the tool	Jan-13
Output:	Experience report for each candidate tool	Apr-13
Output:	Documentation of the changes to each tool (if any)	Apr-13
Output:	Evaluation of the tools against the WP2 requirements	May-13
Output:	Decision on the final tool choice(s)	Jun-13

### 1.3 Identify the tool platform

There is a distinction between tools (Section 1.2) and tool platform: The tools are the core that processes the languages, and typically also have an editor. The tool platform is language independent, but provides mechanisms to integrate various tools. For example, Eclipse is a tool platform. The Java Development Tools (JDT) are an extension to Eclipse that allows working with the Java programming language.

As the toolchain will consist of many tools that must work together seamlessly, it should be analysed independently from the tools. A tool will typically suggest a certain tool platform. The aim of this task is the identification of a tool platform for each candidate tool from Section 1.2. In particular, the platform should provide an suitable to interface (e.g. XML or JSON definition) to facilitate data exchanges between the various tools of the toolchain.

Several levels of interoperability can be supported by the toolchain. For instance, in Eclipse, the most basic level provides support for plugins (OSGi); that is the ability to have several tools inside a common platform. In the project, we will also need support for interoperability at the data level, which can be found in the Eclipse world with the Modeling Framework; that is the ability to have tools that converse/interact using a standardized (low-level) syntax. Finally, we could also require interoperability at a “semantical” level—for example with support for linking objects in different modeling languages—or ask for more basic services, such as serialization and versioning. We believe that, for the project, we at least need interoperability at the data level. Please note that this example is indicative and does not represent a technology decision.

As with modelling tools, tool platforms will be evaluated against WP2 requirements, including in particular compliance against EN50128 constraints.

Input:	List of target platforms, based on the tools being evaluated (1.2)	Nov-12
	Those WP2 Requirements that are sufficient to evaluate a target language	Dec-12
Output:	Evaluation of each tool platform against WP2 requirements, independent of target tool	Mar-13
Output:	Evaluation of tool platform in the context of specific target tools	Apr-13
Output:	Selection of Tool Platform (and reasoning)	May-13



## 2 Supporting Tools Analyses and Recommendations

The languages and tools of the core chain have to be complemented to support a number of activities that are crucial for the project. The purpose of this task is to compose a list of tools which may be used in performing or supporting activities, analyse their suitability and propose a selection for inclusion in the openETCS tool basis. This has to be coordinated with

- the core tool chain selection (suitable tools should be available),
- the definition of the overall development process, as that defines the steps to be performed and
- the safety analyses to conclude if the proposed tools are suitable to fulfil standard criteria for a certifiable tool chain.

The flow of information in the coordination is bidirectional, e.g. the process steps should ideally be tailored towards realizability with good support by available tools. Techniques complementing the tool-supported steps must be included in the process description.

### 2.1 Identify and define the supporting tools and methods

The goal of this subtask is to identify the supporting methods and tools that could complete the tool core chain to achieve the WP2 requirements.

The following activities have already be identified and detailed in the following sections :

- Model Transformation, code generation,
- Requirement traceability
- Validation and verification
- Safety analyses Tools

Input:	WP2: List of suitable languages (based on State of the Art Analysis)	Nov-12
Input:	WP2: Small subset of ERTMS requirements that is representative	Nov-12
Input:	Those WP2 Requirements that are sufficient to evaluate a target language	Jan-13
Input:	WP2: Definition of the process	Jan-13
Input:	WP3a: Decision of the final choice for the core language	May-13
Output:	Evaluation of supporting tools and methods against the WP2 requirements and task 1 choices	June-13
Output:	Decision on the supporting tools choice(s)	June-13

### 2.2 Model Transformation and Code Generation

Analyse model transformation techniques and tools in order to refine the specification from one description level to another.

Analyse the code generation strategy.

Input:	WP2 Requirements	Jan-13
Input:	WP3-a Decision on the final language choice	May-13
Output:	Model Transformation Strategy	Sept-13
Output:	Code Generation Strategy	Sept-13

### 2.3 Requirement traceability

Requirements elicitation is the first step of the process, but we can decide that requirements are already elicited in the subset-026, or selected in WP2. However, how to manage the requirements traceability and how to deal with coverage and traceability of the requirements on models. Thus this activity is a main challenge in developing critical and complex systems.

Input:	WP2 Requirements	Jan-13
Input:	WP3-a Decision on the final language choice	May-13
Input:	WP2 Designer Wishes and Requirements	March-13
Output:	Captured and organised designer requirements	Sept-13

### 2.4 Validation and Verification Tools

In the development of a safety-critical rail system like the ETCS OBU, several kinds of verification and validation activities have to be performed. On the one hand, the relevant standards require the verification of the *safety aspect* of every major design step. They also list constraints on methods (and tools) which may be used for these purposes. On the other hand, any viable development process for a complex real-time system like the OBU needs early validation of design artefacts. This concerns for instance functionality and real-time properties beyond the safety-related issues. The formalisation of design artefacts which comes with a model-based process offers the possibility to employ tools to a substantial degree for these activities.

Validation and Verification activities have different goals and methods :

**Validation** aims to prove that the build software fulfils the user needs, in particular with respect to safety and quality requirements.

**Verification** aims to prove that the build software fulfils the requirements of that phase with respect to completeness, correctness and consistency.

The choice of V&V methods depends on the kinds of properties which are treated.

**Functional Properties.** First, we have to show that all of the components of the system behave as they are intended. This includes at least proving that low-level requirements meet high-level requirements at each stage of specification, and in turn that the code meet these low-level specifications.

One possibility is to use a completely certified toolchain up to code generation, some formal methods (for example the correct-by construction approaches) have shown their effectiveness in many industrial cases. Failing that (*i.e.* if not all the code is generated and/or if the generation

cannot be trusted), we will need to have formal specifications and to verify the code against them. Hoare logic-based tools seem like a good approach in this case, as well as of course test cases generation according to a suitable coverage criterion.

Secondly, we have to show that the high level requirements have been completely specified. This is usually achieved by coverage technics.

**Non-Functional Properties.** Such properties include all aspects that are related to the nominal behavior of the components. In particular, this concerns the following points:

- Schedulability and Worst Case Execution Time (WCET)
- Data dependencies between outputs and inputs
- Modelling and process requirements for SIL4 systems

Schedulability can be done on the models, based on hypotheses for the WCET of single tasks, but verifying these hypotheses require a full knowledge of the code, the underlying hardware, and the compilation toolchain.

Data dependencies can also be assessed on the models, but as in the previous subsection, some verification on the code itself might be required. Static analysers should be able to handle that.

Traceability items can help to check that process requirements are satisfied, code reviews allowed to verify modelling requirements. It is also required by all standards.

**API and modules interfaces properties** A main part of the V&V activity is to check that all components of the system (software and hardware components) can be integrated together.

The first step is to integrate the software components, depending of the choice of the modeling approach formal proof or testing activities could be used to achieved the V&V.

The second step is to integrate the software and hardware parts. Dynamic testing with test automation tool is currently the more convenient approach to validate this integration.

**Safety properties.** This must ensure that components are always able to perform their work. It includes in particular:

- Absence of runtime error (again if the code generation does not guarantee it by construction. Static analysis can be employed there also).
- Fault tolerance against unexpected input (safety analyses can raise this unexpected input).
- High-level safety requirements (formal reasoning, test or simulation can show safety requirements are satisfied).

Input:	WP2 Requirements	Jan-13
Input	WP3b formalized API	Mar-13
Input:	Those sufficient elements of WP4 V&V Plan to evaluate tools	Mar-13
Output:	V&V tool choice	June-13
Output	Test automation tool choice with experience report	June-13

## 2.5 Safety analyses Tools

The purpose of this subtask is to propose a list of tools to support safety analyses activities. This task is to coordinate with the requirements issues of WP2 and the needs of WP4 in regards of safety cases definition.

Examples of interesting tools deals with fault-trees and FMEA generation, that can be based on formal methods (automata, pre-post condition logics,...)

Input:	WP2 Requirements	Mar-13
Input:	Those sufficient elements of WP4 Safety case to evaluate tools	Apr-13
Output:	safety analyses tools choices	Sept-13

## 3 Define and Develop Tool Chain

This subtask defines and develops the tool chain and the infrastructure enabling its evolution and maintenance. First of all, a "make or reuse" decision about the components of the tool chain has to be made. Then a common development infrastructure has to be defined or chosen in order to integrate all the tools (Eclipse like infrastructure). Finally, the subtask achieves the development and the integration of the tools.

### 3.1 Overall Tool Architecture

Once language, method, tools and tool platform have been identified, the architecture can be defined using that as the foundation. The architecture also contains the specification of tool interaction mechanisms (model changes might trigger code generators, model or code changes might trigger regression tests etc.)

Note that this may not be that much work: A tool platform like Eclipse essentially defines the overall architecture already. Further, using an agile approach, it is perfectly acceptable if the system changes over time (i.e. APIs change, etc.), as long as the proper mechanisms are in place, like automated testing.

### 3.2 Development Infrastructure

To allow robust distributed development, care must be taken in setting up a functioning infrastructure. This includes

- a continuous automated build system,
- mechanisms to upgrade tools in the platform,
- mechanisms to add tools to the chain at a later stage (without breaking compatibility),
- tool chain documentation system.

The effort for this must not be underestimated.

### 3.3 Decomposition and Distribution of work

Another major task is the robust decomposition of the tool chain and its distribution and tracking of the various components. Specifically, robust integration tests and version management for the tool chain are crucial<sup>3</sup>.

### 3.4 Inputs and Outputs for Section 3

Input:	WP3a, Section 1: Formal Model representing the sample spec, one for each candidate	Mar-13
Input:	WP3a, Section 1: Decision on the final language choice(s)	May-13
Input:	WP3a, Section 1: Experience report for each candidate tool in the core tool chain	Mar-13
Input:	WP3a, Section 1: Decision on the final tool choice(s)	May-13
Input:	WP3a, Section 1: Selection of Tool Platform	Apr-13
Input:	WP3a, Section 1: Decision on the final (development) method	Apr-13
Input:	WP3a, Section 2: Verification and test tool choice(s)	Apr-13
Input:	WP3a, Section 2: Captured and organised designer requirements	Sept-13
Input:	Bugs report and feedback from project partners (All WPS)	continuously
Output:	Specification of tool interoperability mechanisms	July-13
Output:	Specification of core and support tool chain architecture and its embedding into the platform	July-13
Output:	Infrastructure evolution strategy	Sept-13
Output:	Tool chain qualification test suite	Oct-13
Output:	Tool chain maintenance and debug	continuously

## 4 Develop Open Source Ecosystem

The goal of this task is the development of an open-source ecosystem for the toolchain under development and all its components as well as for the implementation of the ETCS system. This ecosystem defines the license model to be used, the project infrastructure, usage and contributions of existing open-source projects and the process to coordinate the development efforts from different partners. Additionally as part of this task, suggestions and guidelines for the projects infrastructure are developed. The goal of the ecosystem is to facilitate the collaboration of the industrial partners and enable long-term maintenance of the outcome of the development. For the ecosystem, well-established open source ecosystems, such as the Eclipse ecosystem shall be used as templates.

As the development of an ecosystem needs to react on the on-going project and adapt to needs, the ecosystem has to evolve over the project duration. As a first output, this task will produce initial proposals for the license model, the process, as well as the infrastructure. Subsequently, these proposals can be adapted based on the feedback from all project partners.

<sup>3</sup>The related activities are often called *tool qualification*.

As the license model and the open source process have high impact on the overall project, the task will only propose solutions. These solutions are based on requirements given by WP1. Final decisions in this area have to be made by WP1.

Input:	Feedback from project partners (All WPS)	continuously
Input:	WP1: Requirements for the open source ecosystem	continuously
Input:	WP1: Legal decisions	continuously
Output:	Initial proposal for license model	Aug-13
Output:	Initial proposal for open source development process	Aug-13
Output:	Initial proposal for infrastructure and tools	Aug-13
Output:	Adaptation of license model	continuously
Output:	Adaptation of open source development process	continuously
Output:	Adaptation of infrastructure and tools	continuously