openETCS

ITEA2

INFORMATION TECHNOLOGY FOR EUROPEAN ADVANCEMENT
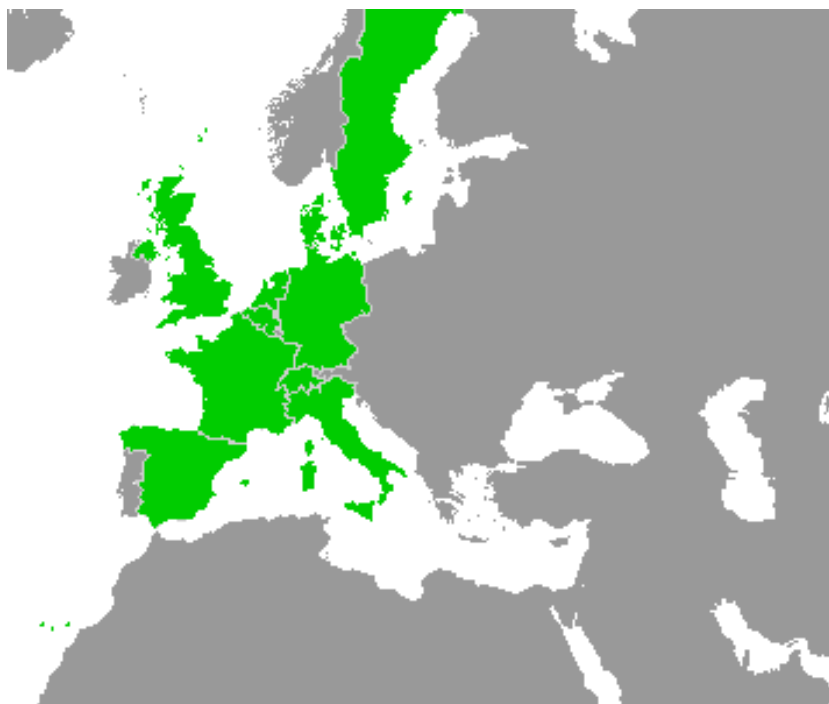
**ITEA2 Project**
**2012 – 2015**

Work-Package 3a: "ToolsChain"

# openETCS Toolchain WP Description of Work

**A template for short document. Adapted from report template.**

Michael Jamstram and WP3a participants                                October 2012

# openETCS Toolchain WP Description of Work

**A template for short document. Adapted from report template.**

Michael Jamstram

WP3a leader

WP3a participants

OpenETCS

Description of work

Prepared for ITEA2 openETCS consortium
Europa

**Abstract:**  This document contents the description of work planned for WP3a.

*openETCS*-

# Table of Contents

Figures and Tables **Figures**

**Tables**

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. The tool chain must support the following tasks:

*(Systerel) Why to restrict the use of formal methods to verification and code generation ? Formal methods can be use at the earliest steps of the design to validate models and to verify traceability between steps.*

1. Support the writing of the formal ETCS system specifications including modeling languages, graphical or textual editors, version management, . . . .

2. Support the writing of a formal ETCS software specifications (while a system specification describes the *problem*, a software specification describes the *solution*).

3. Support code generation from software specifications.

4. Support execution, debugging and simulation of software specifications.

5. Support test case/test data/test oracle generation and test procedure execution from a software test specification.

*(All4TEC) The software test cases should be scheduled at each step of software development life cycle (e.g.: unit test, integration test, final validation test)*

6. Support the verification and validation of the various artefacts, including the formal ETCS specification against the textual ETCS specification.

*(Syterel) If the code is automatically generated from the software specification, is it necessary to generate software test cases e.g unit test ?*

7. Comply with the EN 50128 requirements to tools.

8. Support requirements tracing across all tools and build steps.

*MPA: should be defined, what are its relation to the system specification that is the requirement for the software specification. It should either be part of it as requirements for the software, or be derived from it*

9. Analyse transitions and cooperations between models (graphical/ textual, System/ software / test,...). Several models could be successively performed from system to software level, and at the software level, at least two models could be necessary one for functional specification and one of design.

*(Systerel) "one" integrated tool chain is not necessary the best solution to have a certifiable approach. Indeed integration is a source of faults introduction.*

10. Provide seamless integration of the tools into one tool chain.

*MPA: We should state that it should allow to design the system/software safety case*

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 standards as CENELEC or 50128 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 specifications.

The core tool 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 modelling languages

The objective of this subtask is the identification of the modeling language (or multiple languages), based on the analysis from WP2, by prototyping. For each candidate, a small subset of the ERTMS specification will be modelled. The languages may have to be adapted in the process. The completed prototypes are subsequently evaluated against the requirements from WP2.

This work is based on merit: If a suitable language is identified, but no partner steps up to model the prototype, it will simply not be considered.

Given that different levels of abstraction have to be addressed in the design phase, several languages may be necessary to handle all design phases. Here, the semantics of the languages is the key point; it shall be accurately adapted to the level of description required in each phase of the design.

The identification and definition will distinguish between

- wide-spectrum modelling languages (e. g., UML, SysML, B,...) suitable for a wide variety of modelling domains,

- domain-specific languages (DSL) designed and optimised for application in a specific application domain only.

For wide-spectrum languages their metamodels[1] will be analysed with respect to their expressive power and resulting adequateness for designing ERTMS models. For DSL candidates the associated meta-metamodels[2] will be analysed with respect to their capabilities to support language extensions that may become necessary for novel releases of the ERTMS specification the in the future.

| | | |
|---|---|---|
| Input: | WP2: List of suitable languages (based on State of the Art Analysis) | Oct-12 |
| Input: | WP2: Small subset of ERTMS requirements that is representative | Oct-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 | Feb-13 |
| Output: | Documentation of the changes to each language used (if any) | Feb-13 |
| Output: | Evaluation of the models against the WP2 requirements | Mar-13 |
| Output: | Decision on the final language choice(s) | Apr-13 |

(Systerel) A template of criteria to select the language and tools should be defined.

## 1.2 Identify and compare existing tools

Corresponding to Section 1.1, the objective of this subtask is the identification of the target tool, based on the analysis from WP2, by using it for the prototyping described in Section 1.1.

---

[1] Recall that metamodels specify the syntax and static semantics of a modelling formalism.

[2] 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.

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

## 1.3 Identify the tool platform

There is a distinction between tool (Section 1.2) and tool platform: The tool is the core that processes the language, and typically also has 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 tool. 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.

(ALL4TEC) an interface facility should be examined to facilitate the data exchange between the tools constituting the tool chain (e.g.: XML definition)

**irit-laas** we could stress that there are several levels of interoperability that can be supported by the toolchain. Taking the example of 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.

| Input: | List of target platforms, based on the tools being evaluated (1.2) | Oct-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 | Feb-13 |
| Output: | Evaluation of tool platform in the context of specific target tools | Mar-13 |
| Output: | Selection of Tool Platform (and reasoning) | Apr-13 |

## 1.4 Identify Development Method, including Elicitation Techniques

Analyse requirement elicitation techniques in order to define a strategy to derive OpenETCS formal model requirements from ERTMS SRS

Without a suitable method, most tools have only limited use. Formalising a specification of the size of ETCS requires a suitable method. The method is probably relatively independent from the language used.

The objective of this subtask is the identification of suitable methods and their evaluation while prototyping.

**Christophe GASTON**  A classical modelling process starts by defining high level models of systems that shall be refined step by steps in order to make implementation choices. From a syntactical point of view, the model transformation techniques described below are good technological candidates. However, at the semantical level, refining a model into a more concrete one requires first to define a refinement correctness relation in order to ensure properties preservation. Moreover, systems that will be considered in the project will be based on concepts of interacting processes (since those systems are distributed). Therefore we need to identify techniques allowing us to start from system level properties (*i.e.* specifying global behaviors) and to deduce what properties each process should satisfy (*i.e.* what behavior should have each process) so that the global cooperation of all processes realize the system properties. In this subtask we will identify potential solutions to that problem.

**irit-laas**  we are also in favor of removing this subtask. One drawback of mandating a specific method for developing the models (e.g. refinement or compositional) is that you could rule out some of the tools identified in Sect. 1.2.

More generally, we believe (based on our previous experiences in other domains) that we will end up with several formalisms, targeting different aspect of the specification. We think that we should keep in mind an approach based on modeling different aspects of the problem with languages supporting different viewpoints and that could work at different level of abstractions. With this kind of approach—concerning the verification of the final artifacts—we will also need to check the compatibility of the semantics of the modeling 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.

| Input: | WP2: List of suitable methods (based on State of the Art Analysis) | Oct-12 |
|---|---|---|
| Input: | Those WP2 Requirements that are sufficient to evaluate a target method | Jan-13 |
| Output: | Experience Report on applying the method while prototyping (1.1) | Feb-13 |
| Output: | Evaluation of the methods against the WP2 requirements | Mar-13 |
| Output: | Decision on the final method | Apr-13 |
| Output: | Documentation of adapted method | ongoing |

(SP) I would suggest to remove that subtask. - This belongs to WP2. - Requirement elicitation technique is not a tool, and moreover - ERTMS SRS (Subset-026) already include the requirements.

(ALL4TEC) Christophe Gaston comments address several previous chapters of the DOW. It could be useful to re dispatch them.

(Systerel) Requirements elicitation is the first step of the process, but we can decide that requirements are already elicitate in the subset-026. However, how to manage the requirements traceabilitty and how to deal with traceability between models and activity is a main challenge in developping critical and complex systems. Thus, we think it is important to define how to support requirement traceability.

## 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 respond to standard criteria for certifiable tool chain.

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

The following activites have to be take in account :

- Model Transformation and code generation

- Validation and verification

- Requirement management

- Safety analyses Tools

### 2.1 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 | Apr-13 |
| Output: | Model Transformation Strategy | Sept-13 |
| Output: | Code Generation Strategy | Sept-13 |

### 2.2 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. One 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 artifacts. This concerns for instance functionality and real-time properties beyond the safety-related issues. The formalisation of design artifacts 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 fulfills the user needs, in particular with respect to safety and quality requirements.

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

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

### 2.2.1 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, 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 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.

(DLR) Some further rewriting of this subsection and the following is needed in order to get a consistent description. Concerning the current text, one could broaden the scope of functional verification, including early design steps.)

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

### 2.2.2 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 analyzers should be able to handle that.

Traceability items can help to check that proces requirements are satisfied, code reviews allowed to verify modelling requirements.

### 2.2.3   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 raisonning, test or simulation can show safety requirements are satified).

| Input:  | WP2 Requirements                                          | Jan-13 |
|---------|-----------------------------------------------------------|--------|
| Input:  | Those sufficient elements of WP4 VnV Plan to evaluate tools | Jan-13 |
| Output: | VnV tool choice                                           | Apr-13 |

## 2.3   Test Automation Tools

Testing   is mandatory for the development of safety-critical systems, because no system is allowed to become operative without experimental evidence. In all situations where formal verification cannot be exhaustive, because the size and complexity of the software or integrated HW/SW system exceeds the capabilities of the formal verification power available, testing or simulation is applied to achieve at least partial verification for the cases considered.

In the openETCS context, testing is required in two areas.

- Software and HW/SW integration testing: the overall test objective is to investigate the correct cooperation between several SW and HW components.

- Component (unit, module, thread, process) testing to investigate (1) their functional correctness and (2) the consistency between models and object code generated from the models. The former use case is mandatory if the model has not already been exhaustively verified. The latter use case is relevant when utilising code generators that have not been validated, so that we cannot rely on the correctness of the model-to-code transformation.

In any case a high degree of automation is desirable to support an effective testing process. Available test tools will be analysed with respect to the following properties.

- Availability in open source

- Suitability for the preferred tool platform

- Support for different test levels (unit, integration, HW/SW integration, hardware-in-the-loop system testing)

- Support for automated test procedure / test suite execution

- Support for automated test case / test data / test oracle / test procedure generation from the preferred type of models selected in WP2 / WP3a.

- Support for automated tracing from requirements to test cases to test procedures to test results and back

| Input: | WP2 Requirements | ??? |
|---|---|---|
| Output: | Test automation tool choice with experience report | ??? |

### 2.4 Capture Additional Requirements

Capture wishes/requirements on how to support the designer in their activities.

| Input: | Designer Wishes and Requirements | ongoing |
|---|---|---|
| Output: | Captured and organised designer requirements | ??? |

### 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 interresting tools deals with fault-trees and FMEA generation.

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

(mj) I see "make" not as an option. Considering the resources available for this project, we will tailor (and extend) an existing tool platform.

(jp) Please review this section with me and check it w.r.t. completeness

### 3.1 Overall Tool Architecture

Once language, 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,

- 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[3].

**Inputs and Outputs for WP3a, Section 3.**

| | | |
|---|---|---|
| Input: | WP3a, Section 1: Formal Model representing the sample spec, one for each candidate | ??? |
| Input: | WP3a, Section 1: Decision on the final language choice(s) | ??? |
| Input: | WP3a, Section 1: Experience report for each candidate tool in the core tool chain | ??? |
| Input: | WP3a, Section 1:Decision on the final tool choice(s) | ??? |
| Input: | WP3a, Section 1: Selection of Tool Platform (and reasoning) | ??? |
| Input: | WP3a, Section 1: Decision on the final (development) method | ??? |
| Input: | WP3a, Section 2: Verification and test tool choice(s) | ??? |
| Input: | WP3a, Section 1: Captured and organised designer requirements | ??? |
| Output: | Specification of tool interoperability mechanisms | ??? |
| Output: | Specification of core and support tool chain architecture and its embedding into the platform | ??? |
| Output: | Infrastructure evolution strategy | ??? |
| Output: | Tool chain qualification test suite | ??? |

## 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

---

[3]The related activities are often called *tool qualification*.

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 taks 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.

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) | continuosly |
|---|---|---|
| Input: | WP1: Requirements for the open source ecosystm | continuosly |
| Input: | WP1: Legal decisions | continuosly |
| Output: | Initial proposal for license model | Aug-30 |
| Output: | Initial proposal for open source development process | Aug-30 |
| Output: | Initial proposal for infrastructure and tools | Aug-30 |
| Output: | Adaptation of license model | continuosly |
| Output: | Adaptation of open source development process | continuosly |
| Output: | Adaptation of infrastructure and tools | continuosly |