openETCS Toolchain WP Description of Work

October 12, 2012

Contents

This Work Package will provide the tool chain that is necessary to formalize the ETCS system specification . The formal specification will be used further for verification and code generation. The tool chain must support the following tasks:

MPA: I think it should be: formalize the system specification and implement the software specification

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

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

3. Support code generation from the software specification.

4. Support execution, debugging and simulation of the software specification.

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

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

7. Comply with the EN 50128 requirements to tools.

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

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

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.

1 Core Tool Chain Analyses and Recommendations

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

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.

(All4TEC) 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.

(jp) extended by test data, test oracle, test procedure

(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)

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

(All4TEC) 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.

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

All4TEC ALL4TEC has already experienced tools at several levels of critical railways system and software developments. In order to verify the work, tools like CORE (Vitech) could be used for system technical specifications elaboration (it is a kind of SysML approach leading to definition of the system functional and physical architectures). In on other way Safety Architect could be used for safety studies on various models at diverse levels of definition (system, software, functional, organic etc.).
The following tool chains have been experienced for software testing: For Unitary Test (UT): MATLAB SIMULINK / DESIGN VERIFIER For Validation: MATELO /TESTAND /VERISTAND

1.1 Identify and define the potential modelling languages

The goal of this task is to identify the modelling languages that could fit to requirements associated to specific need of ETCS design. Those needs may come from specificities of ETCS and also from railway norms, and they will be defined by tasks of WP2 (CEA: which WP2 tasks are we talking about?). Depending on recommandation 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:

(jp) Added distinction between wide-spectrum formalisms (e. g., UML, SysML, B) and DSLs

- wide-spectrum modelling languages 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. Since no language is universal, and thus not able to address all aspects of design needs, the proposed approach is lucky 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 modeling languages that address overlapping viewpoints. There are two problems here. First, when dealing with an heterogeneous specification, we need a common semantical

[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.

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)	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

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.

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 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.

(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 Methods

Even though appropriate languages and tools are identified, without suitable design and validation methodologies, most languages and tools have only limited use. Formalising a specification of the size of ETCS requires suitable methodologies. The objective of this subtask is the identification of suitable methods and their evaluation while prototyping. In particular those methodologies have to take into account ERTMS SRS (Subset-026) requirements.

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

2 Supporting Tools Analyses and Recommendations

The languages and tools of choice have to be complemented to support a number of activities that are crucial for the project.

(CEA): we do not understand why there is such a Section 2. In fact all its content could fit in Subsection 1.2. Th e other way round: we do not see what to write in Subsection 1.2 if Section 2 exists because we necessarily overlap.

2.1 Verification Tools

Old: (CEA): we propose to aggregate in this section several subsection related to the kinds of properties which are treated. We tried to identify subsections following this intuition. The idea would be to talk about technologies (*e.g.* static analysis, model-checking etc...) that could address the verification of each sort of property. (iriit-laas): we agree with the proposal from CEA and their choice of subsections.

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.

The purpose of this subtask is to compose a list of tools which may be used in performing or supporting V&V activities, analyze their suitability and propose a selection for inclusion in the openETCS tool basis. This has to be coordinated with both

- the core tool chain selection (suitable V&V tools should be available), and
- the definition of the overall development process, as that defines the steps to be performed.

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.

Old: (DLR) Verification tools resp. techniques are very important in the development of a safety-critical system like the ETCS OBU. According to the relevant standards (most prominently the EN 50128 of the CENELEC family), every design step has to be verified. Assuming that models will constitute artifacts of the development–and are not just used for explanatory purposes–it is necessary to be able to establish the correctness of each refinement step. Or, to put it differently, the tool chain needs a concept for seamless verification, preferably tool-based, to be fit for its purpose.

I think, this must be taken into account already early in the definition process. Therefore, while it might not be the first thing to consider (without modeling, there is no verification of models), it should definitely not be the last.

(DLR) Which WP is concerned with the global picture, i.e. who takes care of the coherency of the proposed parts of the openETCS ecosystem, who defines the process and assigns potential roles to tools?

Old: (ALLTEC) Notice: CENELEC verification activities cannot be systematically addressed with test and tool. For example, source code review could be required to check safety properties of code. Old: (Stanislas Pinte) I agree with Hardi. In my opinion, the model should include the tests of the model, so that it could be verifiable in a "model-in-the-loop" fashion. It doesn't have to be model proving (that I think belongs more to other WPs) rather model testing. Inside our http://www.ertmssolutions.com/ertms-formalspecs/ approach, we assume the following:

- Model tests are part of the model
- Model should be 100% covered by tests (proved by model coverage reports)
- Toolchain must support developing, executing and debugging tests

Model verification also includes:

- verifying that the model corresponds to the original requirements specifications (in our case, UNISIG Subset-026 BL3). ERTMSFormalSpecs also supports marking model artifacts as "verified" against source requirements.
- verifying that 100% of source requirements are traced against one or more model artifacts (proven by traceability reports)

I would think that such verifications are indeed in the critical path...i.e. if not implemented we shall not be able to have a fully functional model.

2.1.1 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. 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.)

2.1.2 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

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.

2.1.3 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

Input:	WP2 Requirements	???
Output:	Verification tool choice	???

2.2 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.

(jp) Some standards (e. g. RTCA DO178B/C) regard testing as a part of the verification process. I think it is clearer, however, to distinguish between verification in the sense of static analysis, abstract interpretation, formal verification, WCET analysis and testing in the sense of dynamic testing in an explicit way in this DoW

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	???

2.3 Code Generation Strategy

Analyse the code generation strategy.

(jp) Should we move code generation and model transformation to the beginning of this section, since this produces the proper product, while verification & test are support activities?

Stanislas Pinte In my opinion, the code generation strategy should be handled in T3.3 Modelling of ETCS specification, that is part of WP3b.

@Fabien, could you confirm WP3b point of view about this?

Additionally, prototypes should be developed for each candidate modelling language, and for each candidate target language.

C. Braunstein, J. Peleska (Uni Bremen) The code generation should be compatible with a domain framework including, for example, the designated operating system API to be applied in openETCS. This should be decide within the tool chain.
Another issue is that code generation should be covered down to binary code, so that V&V arguments may be applied down to the level of binary code.

A more detailed view of these issues may be find in the PositionPaper directory (Uni_Bremen_proposed_Workflow.pdf).

Input:	WP2 Requirements	???
Output:	Code Generation Strategy	???

2.4 Model Transformation

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

Stanislas Pinte In our product ERTMSFormalSpecs (http://www.ertmssolutions.com/ertms-formalspecs/) we use a single, unified model, that is supporting complete Subset-026 logic, with full traceability. If that approach works, why do we need several specification levels?

irit-laas we propose to merge sections 2.2 and 2.3 since they are basically the same kind of activity. It could have the added benefit to balance the "size" with respect to the verification activities in Sect. 2.1

Input:	WP2 Requirements	???
Output:	Model Transformation Strategy	???

2.5 Schedulability

Analyse schedulability tools.

Input:	WP2 Requirements	???
Output:	Schedulability Strategy	???

CEA: this section could disappear and the content would be dispatched in subsection "'non functional properties" of Section "'Verification"

(ALL4TEC) We agree with the remark on schedulability (e.g.: should be removed).

irit-laas: Should we add something about scheduling "synthesis" to the subsection on schedulability? Maybe the scheduling method/startegy is mandated by the SRS? Same remark for adding WCET analysis to section 2.4.

2.6 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	???

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.

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

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 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. Final decisions in this area have to be made by WP1.

Input:	Feedback from project partners (All WPS)	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