openETCS Toolchain WP Description of Work

September 13, 2012

Contents

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

1. Support the authoring of the formal ETCS system specification.
2. Support the creation 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 generation 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 the language itself, the tool for authoring, as well as the method used for formalising the specification.

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.

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 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.

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.

(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.

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.

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 language and tool of choice have to be complemented to support a number of activities that are crucial for the project.

2.1 Verification Tools

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.

Identify potential verification tools with regard to modelling techniques; verification techniques shall be investigated.

Hardi Hungar 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.

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.

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 Code Generation Strategy

Analyse the code generation strategy.

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.

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

2.3 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?

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

2.4 Schedulability

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

Analyse schedulability tools.

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

2.5 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

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

The second 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, tool and tool platform have been identified, the architecture can be defined using that as the foundation. 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 and more. 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 and distribution and tracking of the various components. Specifically, robust integration tests and version management are crucial.

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.