ITEA2 Project
2012 – 2015

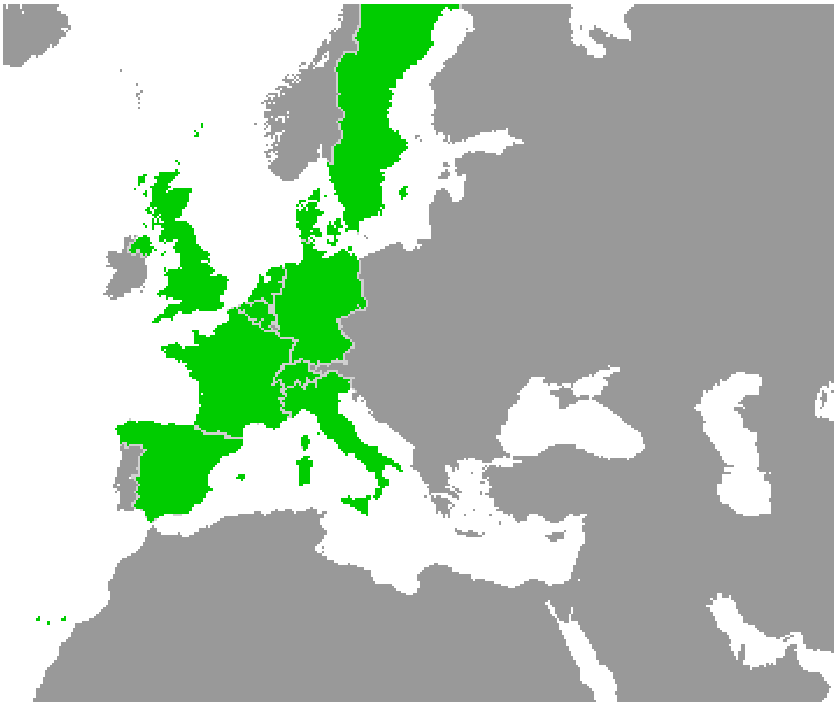
Work-Package 2: “Definition”

# OpenETCS methods

**Definition of the methods used to perform the formal description
Intermediate version**

Marielle Petit-Doche, David Mentré and Mathias Güdemann

May 2013



This page is intentionally left blank

# OpenETCS methods

**Definition of the methods used to perform the formal description**
**Intermediate version**

Marielle Petit-Doche

Systerel

David Mentré

Mitsubishi Electric R&D Centre Europe

Mathias Güdemann

Systerel

Definition

Prepared for ITEA2 openETCS consortium
Europa

**Abstract:** This document give first an introduction to formal methods. In a second part, it proposes the method to fllow during the openETCs project according to the methodology selection.

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

# Table of Contents

1 Introduction .... 4
2 Reference documents .... 4
3 Glossary .... 5
4 Short introduction on formal approaches to design and validate critical systems .... 6
4.1 What is a formal approach? .... 6
4.2 When are formal approaches recommended according to CENELEC standard? .... 6
4.3 Which constraints are required on the use of formal approaches? .... 6
4.4 Which are the benefits to use formal approaches? .... 7
4.5 How to use formal approaches? .... 7
5 Formal approaches for the design and development of a system .... 8
5.1 Contract based approach .... 8
5.2 Model checking of concurrent and synchronous languages .... 9
5.3 Static analysis of software code .... 9
6 Formal approaches for V&V of a critical system .... 10
6.1 Formal approaches for verification .... 10
6.2 Validation of formal approaches .... 10
6.3 Formal methods for safety .... 11
7 Guidelines on the approaches used for OpenETCS .... 11

Figures and Tables **Figures**

**Tables**

| Document information | |
|---|---|
| Work Package | WP2 |
| Deliverable ID or doc. ref. | D2.4 |
| Document title | Definition of the methods used to perform the formal description |
| Document version | 00.02 |
| Document authors (org.) | Marielle Petit-Doche (Systerel)<br>David Mentré (Mitsubishi Electric R&D Centre Europe)<br>Mathias Güdemann (Systerel) |

| Review information | |
|---|---|
| Last version reviewed | 00.01 |
| Main reviewers | Sylvain Baro, Bernd Hekele |

| Approbation | | | |
|---|---|---|---|
| | Name | Role | Date |
| Written by | Marielle Petit-Doche<br>David Mentré<br>Matthias Güdemann | T2.4 Sub-Task Leader | may 2013 |
| Approved by | Gilles Dalmas | WP2 leader | |

| Document evolution | | | |
|---|---|---|---|
| Version | Date | Author(s) | Justification |
| 00.01 | 03/05/2013 | M. Petit-Doche | Incorporation of section on formal methods written by David Mentré |
| 00.02 | 2013-05-16 | D. Mentré | Incorporation of formal reviews in the document |

# 1 Introduction

The purpose of this document is to describe, for the OpenETCS project, the means and methods used to perform the formal description.

However the benchmark activities are not yet achieved in WP7, such the definition of the methods can not yet be done.

For the intermediate version of the document, we propose a description of the benefits of formal methods in the design, development, verification and validation of critical systems.

For the final version, the document proposes the method to follow during the OpenETCS project according to the OpenETCS process and requirements defined in D2.3 and D2.6.

## 2 Reference documents

- CENELEC EN 50126-1 — 01/2000 — *Railways applications — The specification and demonstration of Reliability, Availability, Maintenability and Safety (RAMS) — Part 1: Basic requirements and generic process*
- CENELEC EN 50128 — 10/2011 — *Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems*
- CENELEC EN 50129 — 05/2003 — *Railway applications — Communication, signalling and processing systems — Safety related electronic systems for signalling*
- FPP — *Project Outline Full Project Proposal Annex OpenETCS* – v2.2
- SUBSET-026 3.3.0 — *System Requirement Specification*
- SUBSET-076-x 2.3.y — Test related ERTMS documentation
- SUBSET-088 2.3.0 — *ETCS Application Levels 1 & 2 - Safety Analysis*
- SUBSET-091 2.5.0 — *Safety Requirements for the Technical Interoperability of ETCS in Levels 1 & 2*
- CCS TSI — *CCS TSI for HS and CR transeuropean rail has been adopted by a Commission Decision 2012/88/EU on the 25th January 2012*
- D1.3 – Project Quality Assurance Plan
- D2.1 – Report on existing methodologies
- D2.2 – Report on CENELEC standards
- D2.3 – Definition of the overall process for the formal description of ETCS and the rail system it works in
- D2.6 – Requirements for OpenETCS

## 3 Glossary

**API** Application Programming Interface

**FME(C)A** Failure Mode Effect (and Criticity) Analysis

**FIS** Functional Interface Specification

**HW** Hardware

**I/O** Input/Output

**OBU** On-Board Unit

**PHA** Preliminary Hazard Analysis

**QA** Quality Analysis

**RBC** Radio Block Center

**RTM** RunTime Model

**SIL** Safety Integrity Level

**SRS** System Requirement Specification

**SSHA** Sub-System Hazard Analysis

**SSRS** Sub-System Requirement Specification

**SW** Software

**THR** Tolerable Hazard Rate

**V&V** Verification & Validation

# 4 Short introduction on formal approaches to design and validate critical systems

## 4.1 What is a formal approach?

A *formal* approach is a way to describe system or software that builds upon (i) rigorous syntax and (ii) rigorous semantics.

The *syntax* defines how the system or software description is built and valid. It is usually made through a grammar and a set of additional constraints. It can be textual or graphical.

The *semantics* gives a meaning to each object found in the system or software description. This meaning is given using a mathematical model, i.e., use of mathematical objects attached to each element of the syntax and mathematical rules that define how those objects interacts with other objects. The mathematical models used can be very different from one formal approach to another one. For example the B Method uses the Generalized Substitutions, SCADE relies on the Synchronous language Lustre, etc. One should notice that being able to compile or run a language is not enough to give it some semantics, as this semantics is hidden within the execution/compilation steps. An explicit document should be provided. This document can be informal (e.g. the B-Book) or formal (BiCoq formalization of B Method in Coq formal language).

A *semi-formal* approach is one where the syntax is precisely defined but the semantics is not precisely defined, usually through some English text. Typical semi-formal approaches are the Matlab language or the SysML/UML formalisms.

A semi-formal approach can become formal if its semantics is rigorously defined through a mathematical model.

## 4.2 When are formal approaches recommended according to CENELEC standard?

The use of formal approaches is *Highly Recommended* for SIL3 and SIL4 software according to CENELEC EN 50128:2011.

## 4.3 Which constraints are required on the use of formal approaches?

Each formal approach has some restriction on the kind of software or system it can be applied to. Moreover, each formal approach is specialized in the verification of some kind of property. Therefore a formal approach should be chosen in accordance to the verification objectives.

Moreover, using a formal approach can impact the overall system building process. For example software developed using the B Method follows a specific process and imposes a very specific architecture, very different from designing C software. In the same way, the usage of a formal approach can impose specific resource needs at different phases of the project lifetime. For example, more work on the requirement analysis and formalization phase.

Last but not least, as a formal approach brings its benefits only inside a given boundary, the development process should be designed to transfer these benefits beyond those boundaries. For example, code compilation of a verified source code should be done in such a way as to ensure that the verified properties are kept in the compiled code.

## 4.4 Which are the benefits to use formal approaches?

Several benefits are expected from the use of formal approaches.

The first benefit is to enhance the understanding of the formalized system or software. By using a non ambiguous notation, the designer is forced to clarify his mind. Very often, several design issues or defects are found at this step, and in general, fixing errors at this step is much less costly than in later development phases.

The second benefit is to enable the verification of some properties in an exhaustive way. Therefore avoidance of certain kinds of bugs can be guaranteed. Of course, such guarantee can only be obtained if the formal method is used along some specific way and on a well delimited part of the software and system (for example one cannot guarantee properties on variables outside program boundary).

The third benefit is to allow Correct by Construction software or system building. By verifying properties along the construction cycle of a system or software, one can ensure that some formalized requirements are fulfilled in the final software. For example, one can ensure that some variables stay in well defined boundaries.

The fourth benefit is the ability to easily extend the formalized system or software, by updating the formal description. After such an update, applying the formal verification allows to know precisely which parts are no longer valid and focus development effort on them, without the need to re-verify parts not impacted by the change.

## 4.5 How to use formal approaches?

In the design and development of a system using an approach based on formal methods, there are two orthogonal aspects to consider: at which stage (or stages) in the development cycle the formal approach will be used and how it will be used, i.e., choice of approach, technical realization.

In the development cycle, there are three main stages where a formal approach can be applied:

- Formalization of Requirements
- Design Support
- Implementation Verification

### 4.5.1 Formalization of Requirements

In the System Development Phase and Software Requirements Phase, a formal approach applied to initial requirements can bring clarifications, by enforcing a non-ambiguous meaning for all parties. In case making such a formalization of requirements is difficult, it usually triggers further clarification efforts between involved parties.

### 4.5.2 Design Support

In the Design and Architecture Phase, a formal approach can support the system design and architecture design. In this phase, systematic errors can be detected which can be very difficult and costly or even impossible to fix later.

In combination with a refinement based correct by construction approach, it is possible to have high level properties on the whole system which are refined to sub-properties on the different parts of the system architecture while designing the system. An example of such an approach is the Event-B method.

### 4.5.3 Implementation Verification

In the later phases of the development process, formal approaches can deal with formal reasoning over the actual functional system source code. Depending on the method, this code can be generated from a formal model, derived via a refinement based approach or written manually, annotated with formal properties.

Code generation from a higher level model is in particular interesting, if the generator is qualified and code generation can reduce the required testing of code. A refinement based approach will iteratively add detail to a high level description until a detail level is reached which can be implemented in programming languages, here often translation, i.e., side-by-side creation of refined model and source code is used. And finally it is possible to manually write code which is annotated with properties that can be verified formally (see also Section 5.1. An example for a code generation based approach is SCADE, the B method is based on refinement and formal proof and Frama-C, GNATprove / SPARK are based on source code annotation.

# 5 Formal approaches for the design and development of a system

Very roughly, from an engineering point of view three kinds of formal methods can be used for the design and development of a system:

- Contract based approaches;
- Model checking of concurrent and synchronous languages;
- Static analysis of software code.

## 5.1 Contract based approach

Contract based approaches are based on software (or model) annotation. The software is usually considered state based, i.e. made of a state stored in a set of typed variables. Software is divided in a set of operations (aka procedures, functions, methods, . . . ). To each operation a pre-condition is associated, i.e., a set of conditions that should be guaranteed at operation entrance by the caller of the operation. To each operation there is also a post-condition associated that the operation should fulfill, provided the pre-condition is assumed. In other words, the called operation should ensure the post-condition. The pre and post-conditions are usually expressed using first order logic (and, or, implication, for all and exists quantifiers, . . . ).

In the contract based approach, if all the pre and post-conditions are fulfilled for all possible executions, then we can guarantee that all the operations work well together.

Those kind of approaches are known to be scalable, at the price of sometimes a lot of manual work to properly annotate the software or prove the annotations are correct.

Examples of such approaches are B Method, Event-B, GNATprove/SPARK on Ada language or Frama-C on C language.

### 5.2 Model checking of concurrent and synchronous languages

In this approach, models are based on various textual or graphical formalisms: state based model (like State Machines), data flow equations or Petri Nets.

In a second step, a property is formalized over this model, usually using temporal logic. A temporal logic is usually a first order logic augmented with operators expressing the relationship between events: in the next event, a property is true until another property is true, etc.

Then, model checking techniques (symbolic model checking, exhaustive state enumeration, ...) are applied to check that the expressed property is valid over the model, for all possible executions.

Compared to previous contract based approach, model checking allows to verify more complex properties along the life time of the system. For example, one can express that “something good” will occur in the future after a certain event.

On the other hand, model checking requires a finite state space. In general systems represent an infinite state space and therefore a finite abstraction must be derived for model checking. However model checking suffers from state explosion problem: if the model is not properly designed, it can have too many states and make the exhaustive verification impossible.

Example of such approaches and tools are Design Verifier (used in SCADE), Petri Nets, NuSMV, UPPAAL or SPIN.

### 5.3 Static analysis of software code

Static analysis techniques are inspired by abstract interpretation techniques proposed by Cousot and Cousot in 1977. The main idea is to transform the domain of concrete program variables into a simpler, abstract domain. Then the analysis is done, for all possible execution paths, within this abstract domain. And finally the result of the analysis is put back on the original concrete variables.

Constructing an abstract interpretation is done using specific mathematical approaches (mainly Galois connections) that ensure that the result of the analysis is sound: if an issue is found by the analysis in the abstract domain, it exists in the concrete domains of the variables, i.e., in the real program.

On the contrary, completeness of the approach is difficult to ensure: due to abstraction, the analysis can make some approximation. In such cases the result of the analysis is meaningless: the analysis cannot tell if a verified property is valid or not.

The main advantage of static analysis is that it works on actual, concrete software code, with minimal annotations. It thus integrates quite easily with existing development process, along testing phase for example. And it is highly automated, requesting minimal user intervention.

The drawback of this approach is that it is restricted to certain kind of properties (overflow, underflow, out-of-bound accesses, division by zero). Moreover it does not apply well to all kind of programs.

Example of tool applying such approach are Polyspace, Astrée or Frama-C (with Value analysis plug-in).

# 6 Formal approaches for V&V of a critical system

The various approaches previously presented can be used to check various kind of properties:

- safety properties: ensure the system is safe;
- functional properties: ensure the system works as expected regarding its functional behavior;
- non-functional properties: ensure the system works has expected regarding its speed, capacity, ...

Due to the cost and complexity of formal analysis, use of formal methods in the railway domain is usually focused on ensuring only safety properties. We only consider them is the remaining of this section.

## 6.1 Formal approaches for verification

Ensuring safety properties using formal methods starts in a similar way to classical approaches. A safety analysis will produce the properties that should be ensured to guarantee safe operation of the system.

Usually such safety properties are high level properties (e.g., “two trains do not collide”). In order to be amenable to formal verification, they should be partitioned into properties related to the system state (e.g., “there exists two free blocks between any occupied blocks”, ...) and properties for specific system parts. Several system-related safety properties can be associated to a single high-level safety property. In general it should be verified that as a whole the partial properties imply the high level properties.

Then those properties are checked to be valid, i.e., in any system state a safety property is always true. This can be done with the various approaches presented previously.

If a Correct by Construction or refinement based approach is used, some traditional verification activities like unit or integration tests can be avoided because they are ensured by the formal approaches. In this case the high level property is refined side by side with the system model, iteratively proving the correctness of the refinement steps.

## 6.2 Validation of formal approaches

Proving that a safety property is always valid on a system model does not ensure the property is valid in the real life. Discrepancies can occur between the system model and the real system. Moreover, errors can occur during the formalization of the high-level safety property into a set of system-related properties.

Therefore a validation activity is needed. Validation checks that formalization of properties is correct, as well as all related assumptions.

This is done using non formal techniques:

- Review;
- Simulation and animation;
- Test.

### 6.3 Formal methods for safety

*Comment. proof of safety requirements, static analysis, safety analysis, traceability,...*

## 7 Guidelines on the approaches used for OpenETCS

*Comment. This section will be written for the final version of the document, after the approach and tools tio use during the project will be selected.*