OETCS/WP4/D4.2.2

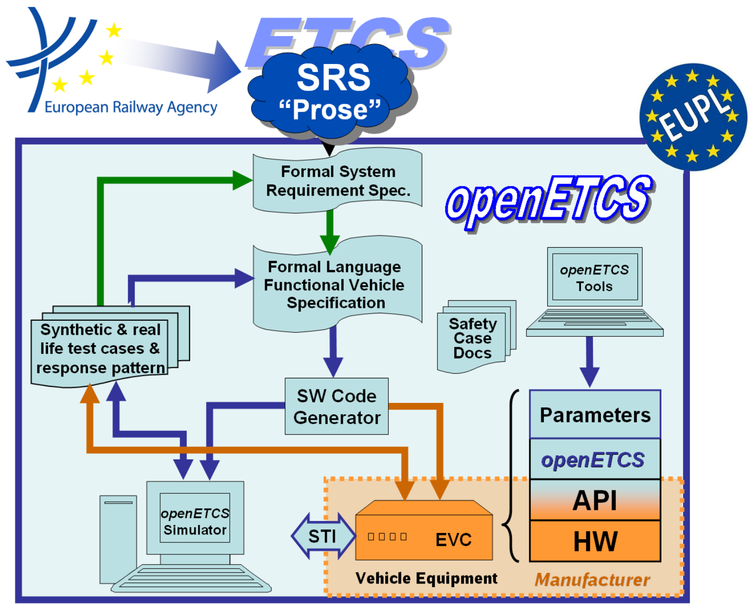
ITEA2 Project
Call 6 11025
2012 – 2015

openETCS@ITEA Work Package 4.2: “Verification & Validation of the Formal Model”

D.4.2.1 1st interim V&V report on the applicability of the V&V approach to the formal abstract model

Ana Cavalli and João Santos

October 2013



Funded by:

Federal Ministry of Education and Research
République Française
Ministère de l'Enseignement Supérieur et de la Recherche
Investissements d'avenir
Région de Bruxelles-Capitale
Gobierno de España
Ministerio de Industria, Energía y Turismo

openETCS

openETCS@ITEA Work Package 4.2: "Verification & Validation of the Formal Model"

OETCS/WP4/D4.2.2
October 2013

D.4.2.1 1st interim V&V report on the applicability of the V&V approach to the formal abstract model

Ana Cavalli and João Santos

Télécom SudParis
9 rue Charles Fourier
91011 Evry Cedex, France

Prepared for openETCS@ITEA2 Project

Table of Contents

Figures and Tables

Figures

Tables

Introduction

To ensure the correctness and consistency of the model and its implementation, the validation and verification has to be performed alongside with the modelling process. Thus these tasks will be performed repeatedly during WP3 and will provide feedback to it.

This document presents the results of the first iteration of verification and validation of the formal model. This was be accomplished by applying the methods chosen in WP4 Task 1 onto the formal model using the tool chain developed in WP7.

This deliverable is a result of the contribution from different partners from the project. The following sections present the contributions of the partners.

1 Institut Mines-Télécom

1.1 Model Based on Extended Timed Finite State Machines

Over the past century, Europe's railways have been developed within national boundaries, resulting in a variety of different signaling and train control systems, which hampers cross-border traffic. In order to increase interoperability for the railway sector, the European Union has decided to adopt and standardize the European Train Control System (ETCS) new methods and tools to verify the reliability of such system need to be elaborated. Therefore, new techniques for verification and testing of ETCS have to be provided. In this project, we are focusing on using model based testing techniques as formal models are proven to be efficient when testing software and hardware products.

In particular, we propose to use finite state machines augmented with continuous variables and guards to represent the most important requirements of the European Train Control System (ETCS) and timeouts. For representing temporal requirements timeouts are used which allow to easily model some critical behavior of the train control system. The above facilities allow describing the basic functioning of the units in this real-time system. At this project step, we are concerned about deriving such model for the ETCS, provide its verification and propose new testing techniques.

The structure of the deliverable is as follows. Section 1.2 contains the specification of the object being modeled, i.e. the European Train system requirements. The formal specification of such requirements is given in the Section 1.3. Methods and tools for model verification are given in Section 1.4 and 1.5, correspondingly. Section 1.6 refers to a testing problem for an ETCS system.

1.2 European Train system requirements

The specification of the ETCS system requirements describes the system behavior as well as a number of functional requirements. As the significance and complexity of these requirements grow rapidly, formal techniques for producing reliable control software become of importance. Such formal methods and model-based verification and testing are among the most promising approaches for increasing software confidence.

To formally describe these requirements, one needs a formalism that takes into account continuous variables (variables related to the train position, speed and acceleration) and also different roles of different actors in the specifications: a) the Radio Block Center (RBC), the train (TRAIN), and the environment itself. The devised formal model must represent critical situations such as a) alarm signals from the RBC; b) external inputs to RBC and trains; c) critical distance between two trains; d) the loss of some messages from/to a train or from/to a RBC.

The following requirements for the train system are considered in the deliverable.

R1. For each train, there is the safety distance d.

R2. If Train$_{id}$ is controlled by the RBC, then the train reports its current position (p), speed (v), and acceleration (a) and the next internal state where the output parameters p, v, a are updated according to the train sensors.

R3. "The input parameter SD represents the safety distance between two trains and is a constant in the model.

R4. Messages between the train and the RBC may be lost. However, the train continues moving and it should automatically decide if it is in a safe position or not.

Those four system requirements are taken into account when deriving a formal train system specification which is represented as a timed extended finite state machine.

1.3 Modeling the European Train System

1.3.1 Formal model for the ETCS

Modeling decisions

There are many models related to ETCS [1, 2, 3, 4, 5]. Most models describe the system behavior using logic formulas and then verify whether these formulae satisfy some safety requirements, such as the safe distance between trains, alarm messages (fire, accidents, etc.) which can come from outside a train and RBC as well as from inside the train. In order to develop a formal model in the ETCS context it is necessary to consider the actors in Figure 1 and discuss the behavioral aspects of an RBC and a train under control and which safety aspects should be taken into account.

1) According to the standards, the ETCS actors communicate in a distributed scenario that is characterized by the absence of a global time, thus, each player has its own clock. Moreover, there are two main safety issues related to time constraints. First, how often the train should report to RBC (position, speed, etc.) and how often the RBC has to send control messages to the train. We do not discuss this issue in the deliverable, since this decision is usually made when verifying the safety of a corresponding logic formula. In our model, this is modeled by discrete time instances when messages may be received/sent. The second aspect is related to the situation when, for some reason, exchange messages are lost. In order to deal with this situation we augment our model with timeout functions. If there is no input before the timeout expires then the player (RBC, a train under control, and/or the environment) has to make their own decision and for the safety issues usually it is the decision to stop the train. Therefore, in our model we synchronize the RBC and train by sending messages with relevant parameters such as the position, the maximum speed, the velocity, or the security point.

2) Another portion of safety issues is related to situations when a train under control moves in an autonomous way: when the train should negotiate with RBC about the safety distance, when the train should be stopped, i.e., we have to check the functional aspects of the system.

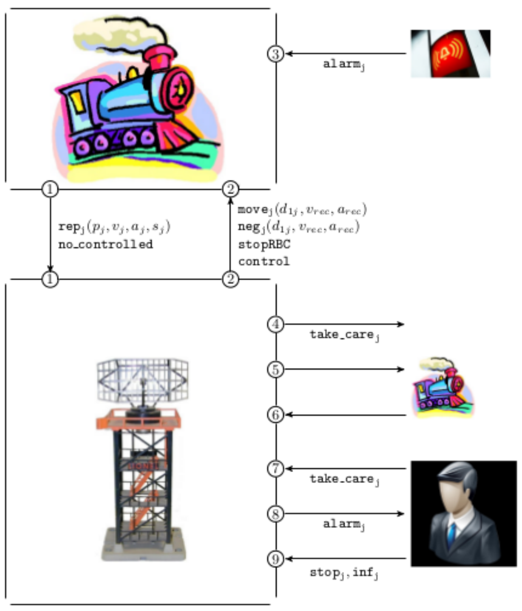


Figure 1. Main Actors in the ETCS.

Some core points can be defined where an actor has different behaviors, and those points can be considered as states in the model. The conditions when an actor moves from one point (state) to another are usually related to some safety distances between trains, respecting alarm messages, etc. Transitions significantly depend on the values of continuous variables such as train position, speed, acceleration, etc.

To solve these issues, we consider an ETFSM as a formal model from which tests for verifying the safety aspects of the developed implementations can be automatically generated. According to our model, there are three actors: a train under control, an RBC that also knows the position, speed, acceleration of the previous train, and the environment. The train under control has four special positions where it has different behavior. These positions are depicted in Figure 2.

Start The initial state is where the train gets a notification message informing that it is controlled by the given RBC. When getting this message the train reports the corresponding data to RBC and moves to another state (moving/stopping).

Moving At this state, the train can get different messages from RBC but almost any message should contain the safety distance according to the position of the previous train or possibly some obstacles reported to the RBC by the environment. If the train is in a safe position it continues moving in an autonomic way. However, if its position is closer to some dangerous point then the train should come to another state and start the negotiation with RBC.

Negot If the train crosses the dangerous point then the train should be immediately stopped. The safety point can be calculated by the RBC based on the data got from the previous train or on some data got from the environment.

Stop The train is not moving, and it waits for an input of RBC in order to start moving again. In our model, we have such an actor as the environment and as far as we know, the idea of modeling the ETCS environment never has been presented. The environment can send alarm messages

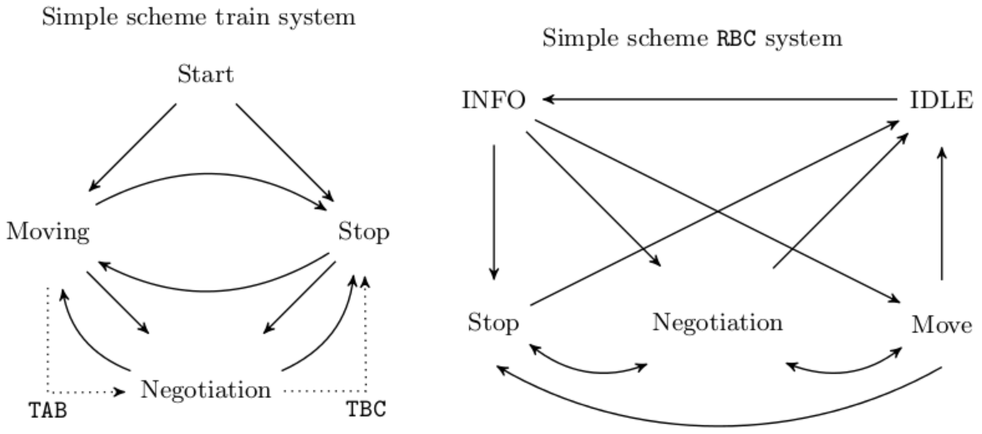


Figure 2. States of the train and *RBC* models.

to the RBC if something happens outside. We also model the train "red button" (accident, fire etc. inside the train) also by the environment and in this case, the train has to immediately report the situation to the RBC. The RBC states are almost the same as for the train but the RBC can get/send messages to/from the environment that usually is some automatic control (another RBC) or some manager in charge. Moreover, based on the collected information (from other RBC, from trains this RBC is in charge of, etc.) the RBC calculates the safety position for a given train and reports this position to the train. Below we present formula of how this safety position is calculated based on the ETCS requirements.

Modeling requirements

In this subsection, we briefly describe the main requirements of the ETCS Level 3. In application Level 3 replaces the line-side signals as well as the trackside occupancy checking devices. The location of the train is determined by the trainside optometry and reported to the trackside radio block centre via the GSM-R radio transmission. In this configuration, train spacing is no longer controlled by the interlocking. However, the latter has to exchange information about the route setting with the RBC. In this level, the trains follow the moving block principle [6], i.e., the current speed and acceleration of a train are dynamically determined by a RBC tracking the train. Trains are only allowed to move when the RBC grants them permission to do so. According to the set of requirements, in this model, we have the first the *requirement R1* which is the critical point of $Train_{id}$ of interest (with respect to the previous train). This point is calculated by the RBC that controls the $Train_{id}$ to avoid collisions. Figure 1 shows the states which are used in the representations of a train under control and an RBC that controls this train.

Figure 1 depicts the actors that are considered in the ETCS, i.e., should be presented in our model, and the exchange messages. The environment actor represents external inputs which may occur in the system, such as an alarm event or a call of an administrator that can interact with the RBC. Thus, the *requirement R2* is also considered in the model.

The RBC analyzes the information obtained from the previous train and returns the critical distance d to the train. According to the rules, the safety distance *requirement R3* is used in the model.

We say that a train at position p and with the critical point d is:

- In a *safe* position if $(d - p) >= 4SD$;
- In a *negotiation* position if $2SD <= (d - p) <= 4SD$;
- In a *stop* position if $(d - p) < 2SD$.

The developed model takes the above expressions into consideration: If the train is at the **Moving** state and it is in a *safe* position then it can remain at this state having the speed and acceleration recommended by the RBC that controls the train. However, if the train progresses and the critical distance is not increased with the same speed, then the train will move to a negotiation position and enter the **Negotiation** state. Finally, if the train is at the **Negotiation** state and the critical distance does not increase then the train moves to a stop position and enters the **Stop** state where the train has zero speed.

Also this Model satisfies the *requirement R4*.

To model this behavior the timeouts $TAB = 4SD/Vmax$ and $TBC = 2SD/Vmax$ are introduced in our model (where $Vmax$ is the maximum speed allowed for the train). If the train is at the **Moving** state and the train does not receive any message from the RBC then after TAB time units it automatically moves to the **Negotiation** state, while if the train is at the **Negotiation** state and does not receive any input message from the RBC during TBC then the train moves to the **Stop** state.

Formal model

We use the following functions when describing the set of transitions of the train and RBC models:

- The function $g(RBC)$ returns the critical point d to the $Train_{id}$;
- The function $f(Train_{id})$ returns the current position (p), speed (v) and acceleration (a) values and the next internal state of the $Train_{id}$ according to the values of the train sensors.

1) **The train**: An ETFSM that describes the internal behavior of the Train$_{id}$ has the following states. State **Start** is the initial state of the train. Any train at this state is not controlled by the RBC of our interest. Once the train is controlled it moves to the **Moving** state.

The RBC checks the position of the train at appropriate time units which usually are very close to each other. If the train crosses the Negotiation point then the train and the RBC start negotiating and the train moves to **Negotiation** state. The dot line TAB denotes a timeout. If the train does not receive any input from the RBC during an appropriate time period (this is discussed above), then the train automatically moves to the **Negotiation** state. The train is at this state if the train has crossed the Negotiation point but did not reach stop position yet; the RBC calculates a new critical position (point d). If the train reaches Stop position then the train automatically should stop (moves to the **Stop** state). Otherwise, it continues moving and will go to the **Moving** state or stay at the **Negotiation** state depending on the value of d and its current position. At the **Stop** state the train has the null speed. The train can come to this state if timeouts are triggered or the current position of the train is very close to the critical point, or the train has received an alarm message from inside the train (modeled as a part of the environment) or a *stopRBC* message from the RBC.

2) **The RBC**: At the initial state **IDLE**, the RBC collects the information of all the trains which are controlled by this RBC and is waiting for the message *take_care* to control a train of interest.

When the message *take_care* is confirmed by the train then the RBC moves to the **Info** state. The **Info** is the state where the RBC waits for a message from the train that contains its position, speed and acceleration and its current internal state.

The RBC sends the message *control(k)* to the train in the **IDLE** state and when $k = j$ the train Train$_j$ is controlled by the RBC. Once getting the message *control(j)* at the **Start** state the train replies with the message *rep(p; v; a; state)* to the RBC. In fact, once controlled by RBC the train replies with this message to any input from the RBC. The message *move(d; V_{max}; a_{max})* sent to the train contains the critical point d (with respect to the previous train or with respect to the next station etc.), the maximum speed V_{max} and the maximum acceleration a_{max} that the train can have.

Depending on the position of the previous train and possibly other conditions, the critical point d is calculated and according to its value the RBC moves to the **Stop** state, to the **Negotiation** state, or to the **Move** state. The **Stop** is a state where the RBC finds out that the train is stopped.

It could happen because the position of the train is close to the point d (the train is in a Stop position) or if any external input alarm occurs. The **Negotiation** is a state that denotes an active exchange of messages between the train and the RBC since the train is not in a *safe* position but did not reach a *stop* position, i.e., the train is in a *negotiation* position. The **Move** state denotes that the train is moving as autonomous as possible.

The RBC sends the message *stopRBC* in order to stop the train. If the train receives this message at any state then the train moves to the **Stop** state.

The message *neg(d; V_{max}; a_{max})* is sent to the train when the RBC knows that the train is at the **Negotiation** state. The environment can send the *alarm* message to the RBC indicating that something is going wrong outside.

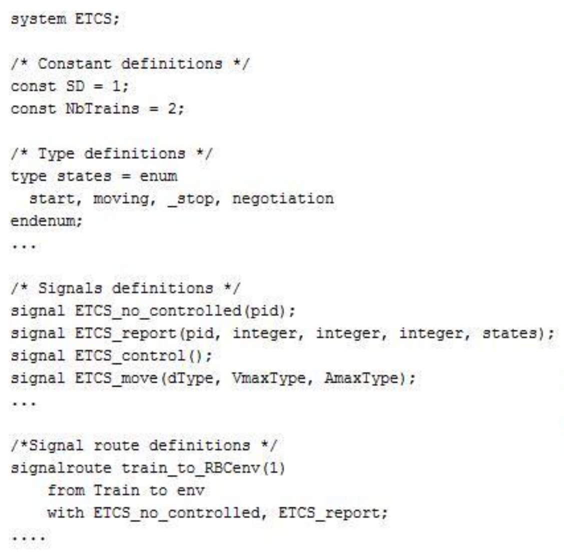
Finally if the RBC gets this message then RBC sends the message *stopRBC* in order to stop this train and enters the **Stop** state. There is a timeout TAB from state **Moving** to the **Negotiation** state, where $TAB = 4SD$ (four times exceeding the safety distance):

*If the train is in the **Moving** state and the train does not receive any input during TAB time units, then the train automatically moves to **Negotiation** state.*

There also is a timeout TBC from the **Negotiation** state to the **Stop** state, where $TBC = 3SD$ (three times exceeding the safety distance) that means:

*If the train is in the **Negotiation** state and the train does not receive any input after TBC time units, then it automatically moves to the **Stop** state.*

1.3.2 Using specification languages to describe the model



```
system ETCS;

/* Constant definitions */
const SD = 1;
const NbTrains = 2;

/* Type definitions */
type states = enum
  start, moving, _stop, negotiation
endenum;
...

/* Signals definitions */
signal ETCS_no_controlled(pid);
signal ETCS_report(pid, integer, integer, integer, states);
signal ETCS_control();
signal ETCS_move(dType, VmaxType, AmaxType);
...

/*Signal route definitions */
signalroute train_to_RBCenv(1)
    from Train to env
    with ETCS_no_controlled, ETCS_report;
....
```

Figure 3. A sample code of the ETCS system specification in IF.

Using XML and Java for model specification

In order to assess the possibilities of the model to represent safety properties a prototype in XML-based language has been implemented that was then automatically translated into Java.

A simulator of an EFSM with timeouts has been developed: given a timed parameterized input sequence, the simulator reproduces the corresponding sequence of outputs. In order to provide a clear representation of the model to the simulator, we depict all components using XML files. In each file, the main component is state and inside each state all its characteristics are presented. Among the possible things, there are transitions that start at this state, internal variables for each state, update functions for these variables and timeouts. Once the XML file is properly defined, it is parsed in order to create an internal representation of the ETFSM and we use JDOM, a Java library for XML parsing. During this phase, all the components are created internally to be used later for the simulation. After this phase is completed, the model is ready to be executed. The simulator itself is implemented in Java, with a different thread in charge of keeping time (issuing an alarm when a timeout has occurred). Also, each component is represented internally by a different object, for example transitions and timeouts are different object types with very distinct characteristics.

Nevertheless, to perform model checking we have used different languages to specify the corresponding TEFSM. One of alternative can be the IF language which allows to efficiently specify train safety properties.

Using IF for model specification

IF is a language based on temporized machines, allowing the description of existing concepts into specification formalisms. A real-time system described using IF language is composed of processes running in parallel and interacting asynchronously through shared variables and message exchanges via communication channels. The description of a system in IF consists in the definition of: data types, constants, shared variables, communication signals and processes. In Figure 3 we represent a short code of the ETCS system in this language.

Using SysML for model specification

SysML (Systems Modeling Language) is a general-purpose graphical modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. It is defined as an open source specification of a subset extension of the UML, i.e., SysML reuses seven of fourteen diagrams of UML and add two diagrams: requirement and parametric diagrams. It seeks to generalize UML for specifying complex systems that include non-software components such as information, processes, hardware, personnel and facilities.

In this project, we try to use SysML for model specifications. The SysML models are then validated by using SPIN model checker after translating them automatically into Promela.

1.4 Model verification

1.4.1 Deriving safety properties

When performing the verification of the TEFSM model we considered the following safety properties.

Property 1

The train is in the **Moving** state, running at 100km per hour, with an acceleration of 10km per h^2 on a distance of 100km. Due to some external unexpected reasons, the train must stop as soon as possible. Therefore, an alarm from the environment is sent to the train. Automatically, the train should stop and provide a reporting message with its current parameters (state, position, etc.) to the RBC.

Property 2

In this case, the property is the capability of the ETCS system of taking over control if the driver appears to be going too fast.[1] We consider the train in the **Start** state, receiving the message move with a certain speed and acceleration.

Property 3

A train is in the **Moving** state, running with a speed of 100 km per hour, with an acceleration of 10 km per h^2, situated somewhere at 10 km from the reference point. The safety distance (SD) between two ETCS trains is of 300 km. The train receives the signal move with the maximum speed of 105 km per hour and with an acceleration of 15 km per h^2 for the next 900 km.

Property 4

The property is related to transitions with time-outs. We consider a train in the **Moving** state, moving with a speed of 100 km per hour, with an acceleration of 10 km per h^2. The safety distance is of 300 km and the maximum speed allowed is V_{max} = 105 km per hour.

The communication between the train and the RBC is lost, so the train should take an appropriate decision on the next state, after an appropriate period of time.

[1]This property represents the situation that caused the Spain train accident

We further discuss different verification techniques that can be used to check the above properties based on the TEFSM.

1.4.2 Proof technique

A proof is a demonstration that if some fundamental statements (axioms) are assumed to be true, then some mathematical statement is necessarily true. As mentioned in the requirements document produced by WP2, as much as possible, formal proof would then be used to prove that the OpenETCS model never enter a Feared State, as long as the other subsystem (RBC, communication layer, etc.) fulfill their own safety properties (axiom describing the environment). Such theorem proving helps to increase our confidence on the specified model. The proof techniques should be integrated in the selected tool chain. In order to use formal proof to verify if the SFM (Semi-formal model) and FFM (fully formal model) comply with the safety and function requirements (cf. R-WP2/D2.6-02-058), the properties to be proven have to be identified and described. There will be a set of axioms that will describe both functional and/or safety properties of the system. The choice of axioms describing functional and/or safety properties will be provided by safety analysis in an independent way from approaches used to specify, design, validate or verify. It must be noted that the model obtained from the Subsystem Requirements Specification should be verified in this manner at a first stage.

1.4.3 Model checking

Model checking is an automatic technique for verifying finite-state reactive systems. Using such techniques one could automatically check if the model specifies most of the requirements of the system, such as the important safety properties described in Task 4.4. Similar to proof techniques, in order to use model checking to verify if the SFM (Semi-formal model) and FFM (fully formal model) comply with the safety and function requirements (cf. RWP2 /D2.6-02-058), the properties to be proven have to be identified and described. To implement the use model checking, it is mandatory to specify the model using finite-state reactive systems, and they should also provide an intuitive way to express the properties to be model checked. The language for describing a formal model for which corresponding model checkers exist should be selected and the set of critical requirements to be verified need to be clearly identified. The proposed model checking techniques should be supported in the selected tool chain. As mentioned above, we are using XML-based and IF-based specification to perform the model checking of train safety properties.

1.4.4 Simulation

When using model checkers the criteria for the model to be safe all the safety properties should be checked. The latter is impossible, since the number of safety properties is infinite and thus, only some of them can be checked through the above step.

For this reason, as a complementary approach, testing is commonly used. If the corresponding model respects safety requirements, i.e., only expected outputs are produced to applied input sequences, to some extent, there is a confidence that the models is safe. In order to derive 'good' tests a formal model should be involved. It is known that only the FSM model where each input is followed by a corresponding output allows automatic deriving finite tests with the guaranteed fault coverage where the races between inputs and outputs can be easily avoided. Many authors for deriving finite tests with the guaranteed fault coverage turn their model to some kind of an FSM (see, e.g., [7, 8, 9]).

As for simulation, the artifacts should provide means to execute the model. The simulator must be automatically generated, so that, when run against test scenarios (inputs/outputs for the model), we may conclude whether the model follows the specification or not. In particular, it is important to define test scenarios for the safety critical properties. Since, the developmentwithin openETCS has to the goal to reach the CENELEC EN 50128 SIL 4 standard, it is highly recommended (cf. SIL 4) that the simulation needs to cover all states, transitions, data-flow, and paths in the model. It would also be desirable to include graphical representation of the simulation/model and also provide a report of the visited components as specified by CENELEC EN 50128 SIL 4. CENELEC EN 50128 SIL4 also advocates to perform tracing. Being able to trace the requirements that are met during a simulation is also advisable to allow simple requirement coverage.

1.4.5 Other Methods

Reviews, Inspections, static analysis and walkthroughs, mostly manual techniques, are also to be considered for the verification of models.

1.5 Existing software tools for model verification

1.5.1 SPIN model checker

SPIN (Simple Promela INterpreter) [10] is an open-source software too1, written in C, for simulating and verifying concurrent systems. The communications, via message channels, between concurrent processes of system can be synchronous, i.e., rendezvous, or asynchronous, .i.e., buffered. Models of systems to be analyzed are described by Promela (PROcess MEta LAnguage). In simulation mode, the model is executed step-by-step to familiarize with the behaviors of system. In verification mode, the model is exhaustively explored in order to show that it satisfies some critical properties, e.g., deadlock-free, violated assertions, or general correctness requirements given in Linear Temporal Logic formulas. In this mode, SPIN generates a C program that constructs an implementation of the model-checking algorithm for the given model. The program is then compiled and run to get the result.

As SPIN is proven to be efficient model checker we further plan to try a model checking approach using Promela specification. For this purpose, we will derive a Promela specification from of the TEFSM and perform the model checking of safety properties. Such safety properties will be specified as assertions, and corresponding violations will provide the counter-examples if the model will be invalid.

1.5.2 Using JPF model checker

Java Pathfinder [11] is a system to verify executable Java bytecode programs. The core of JPF is a Java Virtual Machine that is also implemented in Java. JPF executes normal Java bytecode programs and can store, match and restore program states. Its primary application has been Model checking of concurrent programs, to find defects such as data races and deadlocks.

As in this project, the XML-specification of the train system is automatically converted into Java code we also consider to use this model checker in the future. In this case, safety properties will be specified as Java assertions and will be added to the corresponding Java implementation.

1.6 Testing European Train System

1.6.1 Adaptive scenario

In this section, we present the notion of testing and discuss a testing scenario for our model. We assume that the RBC and the train are tested separately, i.e., when testing the train software the RBC is replaced by a tester that sends inputs to the train and checks whether the produced outputs are expected. Moreover, we consider adaptive testing, i.e., a test case is represented as an acyclic ETFSM [12, 13] where only one timed input is defined at each intermediate state. We start at the initial state and apply an input defined at the state to an IUT. Based on the produced output the FSM R that represents a test case will move to another prescribed state where another input can be specified and exactly this input will be applied at the next step. If the observed trace is a trace of the specification TEFSM then we say that the IUT passed a test case. Otherwise, the IUT fails the test case, i.e., the IUT does not conform to the given specification. We illustrate our approach using the following testing scenario (Figure 4):

If there is no signal from RBC (lost messages) for an appropriate period of time then a train must stop itself.

In Figure 4 are represented the following five steps:

1. The tester (simulating the RBC functioning) starts with sending the message control to the train. The 0 value in the initial state means that the RBC does not wait anything to send this message.
2. Depending on the internal state of the train, the tester moves to different states. In order to simplify the example let us consider that the train always answers this message.
3. At these states the tester waits for the appropriate number of time units $TAB + TBC$ or TBC when there are no messages sent to the train.
4. Then the tester checks the state of the train by sending the control message again. If the train is in the **Stop** state then the tester returns *Pass*, otherwise it returns *Fail*. Finally, having the train at the **Stop** state the tester forces it to move with the maximal permissible speed and acceleration and returns the critical distance d.

In the same way, other testing scenarios for checking safety properties may be considered. For example, we can derive a test case that checks whether a train stops when being close to the critical point d, a test case that checks whether a train changes the Moving state when crossing the critical point d, a test case that checks whether a train speed (acceleration) does not exceed the permissible maximum, etc. Finally, it is worth mentioning that when we are testing whether a train stops when reaching a critical point or whether the train respects RBC requirements about its speed and acceleration we must use continuous variables.

1.6.2 Test generation based on IF representation

In this Section we present how to automatically extract a set of tests from the specification. To do this, the behavior of the train process is described in the IF language. We have identified a set of basic requirements and we can describe them as properties in IF. Based on these properties, the validation and verification of the formal specification are carried out using the IF toolset TestGen-IF which automatically generate a set of adaptive tests.

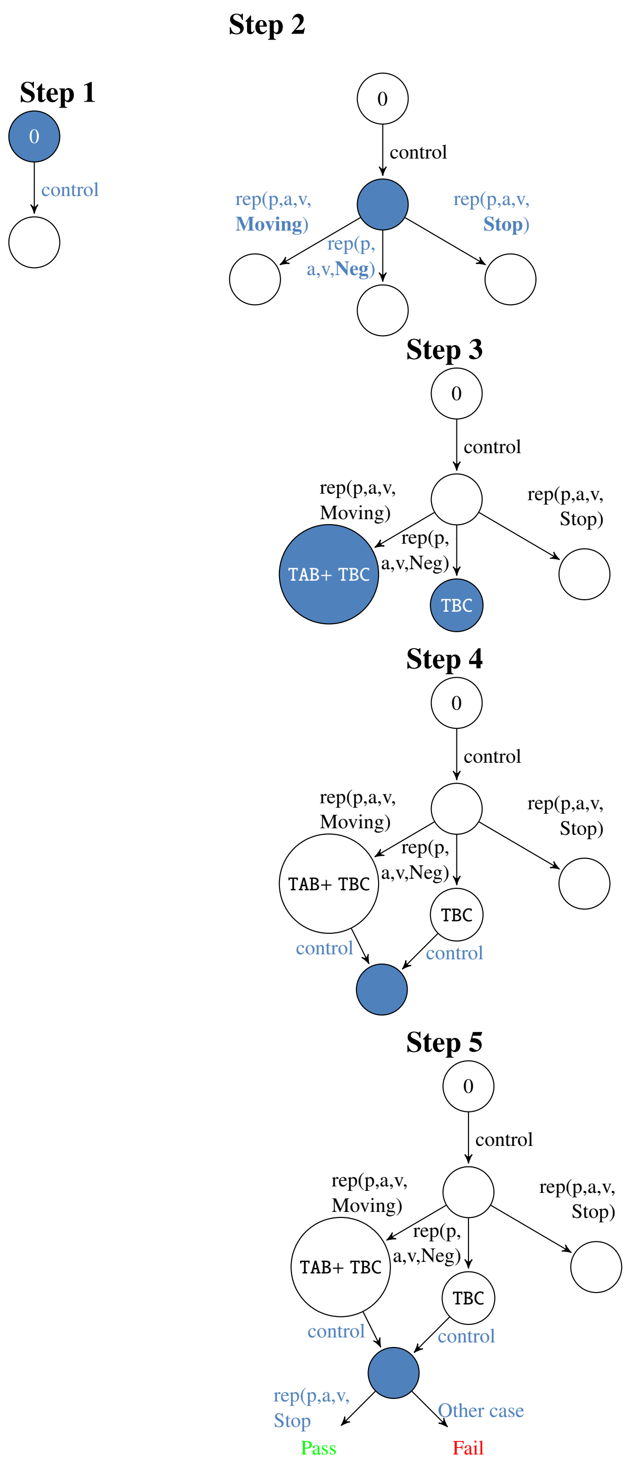


Figure 4. Testing scenario.

We present a short overview of the result of applying this tool in a requirement related to Property 1 (Section 1.4.1). The set of test objectives associated to this scenario, noted as OBJ(1) is formally described in Figure 5.

Let us note that, according to the system specification, the property OBJ(1) is correct in any of the states of the train model. Therefore, at any state, the train should stop when receiving an external alarm input.

```
#Test-case for the train in the Moving state
?ETCS_control{}!ETCS_report{{Train}0,10,100,10,start}
?ETCS_alarm_to_train{}!ETCS_report{{Train}0,10,
 100,10,moving}
?ETCS_control{}  !ETCS_report{{Train}0,10,0,0,_stop}
#Test-case for the train in the Negotiation state
?ETCS_control{}!ETCS_report{{Train}0,10,100,10,start}
?ETCS_move{900,105,15}!ETCS_report{{Train}0,10,100,
 10,moving}
?ETCS_alarm_to_train{}!ETCS_report{{Train}0,10,100,
 10,negotiation}
?ETCS_control{}!ETCS_report{{Train}0,10,0,0,_stop}
```

Figure 5. Set of test using TestGen IF tool.

Different test scenarios are performed with respect to safety properties discussed in Section 1.4.1. As a future work we plan to verify the fault coverage of these tests by executing Java simulator against corresponding traces.

Results

In this deliverable, we have provided a formal model for the requirements of the European Train Control System using finite state machines augmented with continuous variables (train position, speed, acceleration) and time constraints. The model is a very close representation of the system specification provided by the standard.

We have discussed different model checking techniques to verify safety properties for corresponding TEFSM representing the ETCS. To efficiently perform such model checking we use different specification languages, namely, XML and IF languages. We have proposed a technique of the ETCS adaptive testing w.r.t. test scenarios written as train safety properties.

As a future work, we plan to use different model checkers and perform experiments with the model being derived. We plan to use SPIN and for this purpose we will specify the TEFSM in the SysML language that will be further translated into Promela. Meanwhile, we plan to use the JPF model checker to efficiently utilize the Java train simulator that has been developed during the first part of the project.

2 TWT GmbH Science & Innovation

One of our activities regarding model verification is the modelling of the behavioral part of the ETCS (i.e., the procedures described in Subset 026, Chapter 5). The models are then used to validate the specification, to support the modeling using Scade and the verification of Scade models on a higher[2] level of abstraction.

2.1 Achievements

So far, we have modeled parts of Subset 026-5. During this process and with the resulting model, we have conducted a first validation of the specification.

[2]In comparison to Scade models

2.1.1 The Model

Until now, we have modeled the following five procedures of Subset 026-5:

- Start of Mission (Subset 026-5.4)
- End of Mission (Subset 026-5.5)
- Shunting Initiated by Driver (Subset 026-5.6)
- Override (Subset 026-5.8)
- Train Trip (Subset 026-5.11)

As a formal model, we use *colored Petri nets* (CPNs) [14], an extension of classical Petri nets [15] with data, time, and hierarchy. CPNs are well-established and have been proven successful in numerous industrial projects. They have a formal semantics and with CPN Tools [16], there exists tool support for modeling CPNs. Moreover, CPN Tools also comes with a simulation tool and a model checker, thereby enabling formal analysis of CPN models.

We focus on modeling the *system behavior*, that is, the control flow of the on-board unit and the interplay with its environment (e.g., the driver and the RBC). Figure 6 depicts the CPN representing the highest level of abstraction. It shows the decomposition of the overall system into the on-board unit and its environment: the driver, the RBC, the RIU, the STM, and the GSM module. Each component is modeled as a subpage (i.e., a component). Graphically, a subpage is depicted as a rectangle with a double-lined frame. Furthermore, the model shows through which message channels and shared variables the on-board unit is connected to its environment. A channel or shared variable is modeled by a place which is graphically represented as an elipse. As an example, the driver (i.e., subpage `Driver`) may send a message to the on-board unit (i.e., subpage `On-board Unit`) via the place `msg from driver`, and receives messages sent by the on-board unit via the place `msg to driver`.

Zooming in subpage `On-board Unit` yields the CPN model in Fig. 7. This CPN model has two subpages: Subpage `Start` models the states S0 and S1 of the specification (i.e., Subset 026-5.4) and subpage `Rest` the remaining states. At this level of abstraction, we see on the left hand side seven places (green frame). Each such place models (a part) of the state of the on-board unit, for example, the mode and the train running number. The current model has 689 places, 173 transitions and 1,227 arcs.

Having a more detailed look at Fig. 7, we observe that our model does not represent all variables of the on-board unit as given in the specification and also partially abstracts from data. We abstract from those details, because the model is tailored to formalize the *control flow* of the on-board unit and, in particular, the *communication behavior* with its environment. As a benefit, this abstraction reduces the complexity of the model and improves its understandability. Additional details, such as data and precise message values, can be added in a refinement step.

2.1.2 Validation of the Specification

The modeled procedures have been manually modeled using CPN Tools. Thereby, each element in the model has been reviewed against the respective requirement, as given in the specification. To improve the confidence in the model, in a second step, a person other than the modeler checked

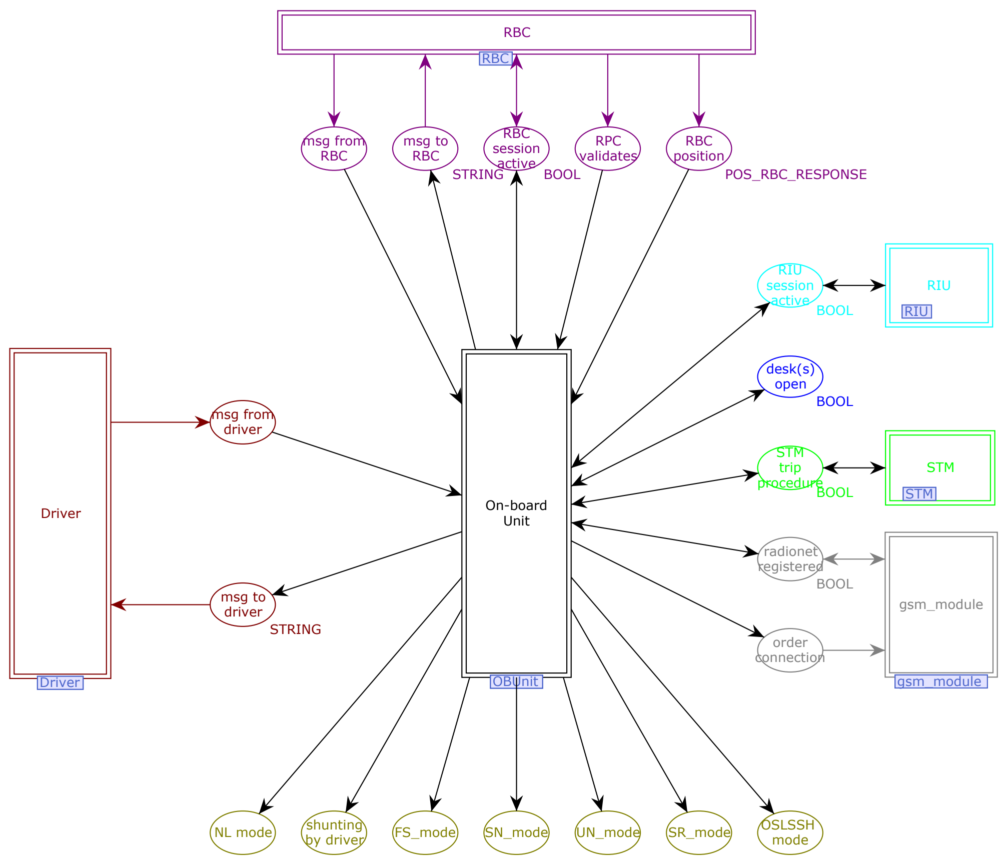


Figure 6. Top level model

the model against the specification. In addition, we used the simulator to check whether the modeled behavior of the CPN matched the intended behavior.

So far, the primary goal of modeling has been to validate the specification. During the modeling we discovered 36 inconsistencies, ambiguities and gaps in the specification which we reported in [17].

2.1.3 Other application of the model

The CPN model can also serve as a *reference model*, for example, to compare and check other models and to generate test cases.

2.2 Next Steps

We shall continue our work by completing the model, contributing to the modeling of (parts of) Subset 026-5 using Scade, and verifying the Scade model. In addition, we are planning to exploit synergies by collaborating with the project partner LAAS who advocate the Petri net model checker Tina [18].

2.2.1 Modeling the Subset 026-5

We plan to model the remaining parts of Subset 026-5, thereby reporting possible additional findings in the specification. The goal is to have a CPN modeling all procedures that are described in Subset 026-5. We also want to compare our model with the (corresponding part of the) ERTMS model [19].

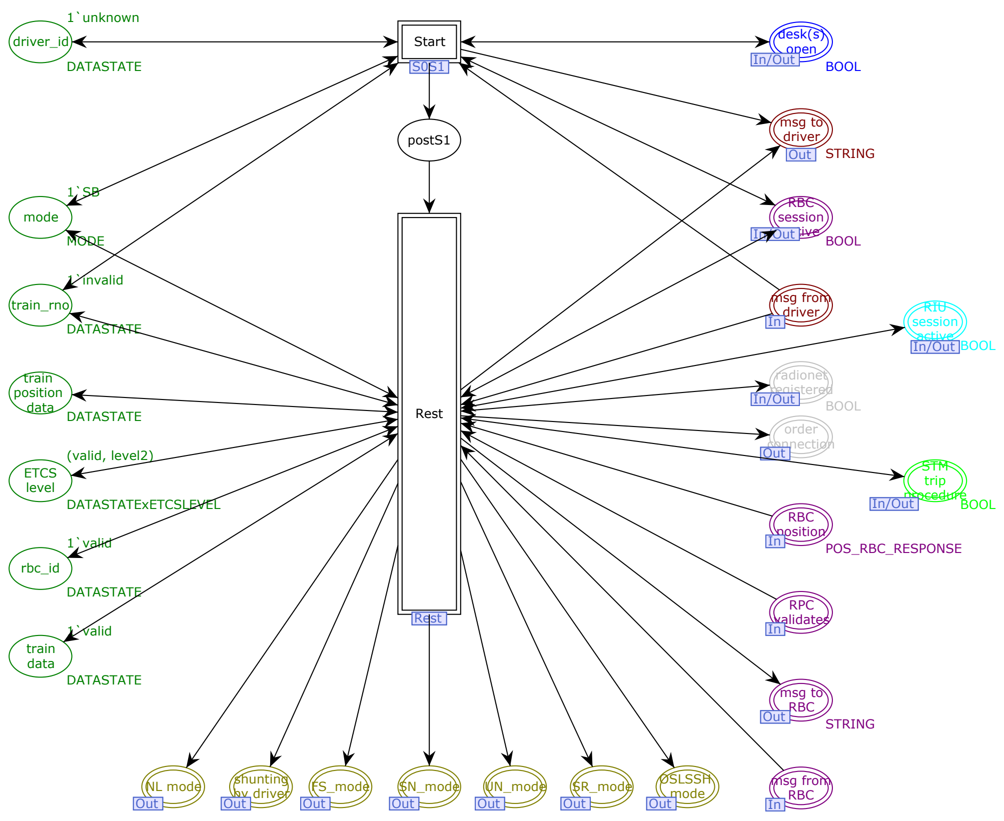


Figure 7. CPN model of the on-board unit

2.2.2 Scade Modeling

As the ETCS will be modeled using Scade, we shall contribute to this modeling process. To use the experience that we gained from modeling Subset 026-05 with CPN Tools, we want to contribute to the Scade modeling of (parts of) the Subset 026-5. The Scade design flow starts with modeling all components and their interplay using SysML block diagrams (with the tool Scade Designer). The resulting SysML diagrams provide a functional and an architectural view. They are similar to the CPN model in Fig. 6. In a second step, the behavior of each block has to be fully modeled on the system level using Scade Suite. As Scade currently does not support state machine models on the level of SysML, the Scade design flow misses an abstraction level. Our CPN model closes this gap and will, therefore, be useful for the Scade modeling.

2.2.3 Verification of the Scade Model

Another task concerns the verification of the resulting Scade model. Recently, researchers reported on complexity problems already for medium-sized Scade models that restrict the verification using the Scade prover [20, 21]. Given the complexity of the ETCS, we assume that we will face similar challenges. To alleviate those complexity problems, we aim to apply the following techniques:

Abstraction

We will apply abstraction techniques on the Scade model to prove safety properties on a higher level of abstraction whenever possible. On the one hand, we can apply Scade contracts to restrict the domain of the input values. This technique is known as environment abstraction. Or, we can transform the Scade model into a model of higher abstraction, thereby using different formalisms such as timed automata, transition systems and Petri nets. We can then use verification tools that are dedicated to the properties of interest and the chosen formalism. We see the chance that our CPN model can be used for this task, too. For example, Uppaal [22] can analyze timed automata, Spin [10] and NuSMV [23] can analyze transition systems, and Tina [18], LoLA [24], and CPN Tools [16] are tools for analyzing (different variants of) Petri nets.

Compositional Reasoning

Another approach is to prove properties for individual components and deduce from it the correctness of a property concerning the entire ETCS. Here we think that we can, in particular, combine our model with the MoRC model [25] and apply compositional reasoning.

Correctness by Design

The two previous approaches support *correctness by verification*; that is, first the model is designed and in the next step it is verified. A different methodology is *correctness by design*. The idea is to model on a higher level of abstraction and to prove that certain safety properties hold. Then the model is iteratively refined. Each refinement step has to guarantee that all properties that hold for the more abstract level also hold in the refined model. The challenge is to find property-preserving refinement rules or a refinement relation between an abstract model and a refined model that preserves the desired properties and to verify that this relation holds.

References

[1] J. Padberg, L. Jansen, H. Ehrig, E. Schnieder, and R. Heckel. Cooperability in train control systems: Specification of scenarios using open nets. *J. Integr. Des. Process Sci.*, 5(1):3–21, January 2001.

[2] A. Zimmermann and G. Hommel. Towards modeling and evaluation of etcs real-time communication and operation. *J. Syst. Softw.*, 77(1):47–54, July 2005.

[3] P. Ammann, J. Offutt, and S. Version. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 1st edition, 2008.

[4] Y. Liu, T. Tang, J. Liu, L. Zhao, and T. Xu. Formal modeling and verification of rbc handover of etcs using differential dynamic logic. In *10th Int. Symposium on Autonomous Decentralized Systems,ISADS '11*, pages 67–72, 2011.

[5] Johannes Feuser and Jan Peleska. Model Based Development and Tests for openETCS Applications – A Comprehensive Tool Chain. In *In Proceedings of FORMS/FORMAT 2012*, pages 235–243, 2012.

[6] A. Platzer and J.D. Quesel. European train control system: A case study in formal verification. In *11th Int. Conference on Formal Engineering Methods: Formal Methods and Software Engineering, ICFEM '09*, pages 246–265. Springer, 2009.

[7] J. Springintveld, F. Vaandrager, and P. R D'Argenio. Testing timed automata. *Theoretical computer science*, 254(1):225–257, 2001.

[8] M. Zhigulin, N. Yevtushenko, S. Maag, and A.R. Cavalli. Fsm-based test derivation strategies for systems with time-outs. In *11th Int. Conf. on Quality Software, QSIC'11*, pages 141–149, 2011.

[9] M. Gromov, K. El-Fakih, N. Shabaldina, and N. Yevtushenko. Distinguing non-deterministic timed finite state machines. In *Joint 11th IFIP WG 6.1 International Conference FMOODS 2009 and 29th IFIP WG 6.1 International Conference FORTE'09*, pages 137–151, 2009.

[10] G.J. Holzmann. The model checker spin. *Software Engineering, IEEE Transactions on*, 23(5):279–295, 1997.

[11] `http://en.wikipedia.org/wiki/Java_Pathfinder`.

[12] A. Petrenko and N. Yevtushenko. Adaptive Testing of Deterministic Implementations Specified by Nondeterministic FSMs. In *23rd IFIP WG 6.1 international conference on Testing software and systems,ICTSS'11*, pages 162–178, 2011.

[13] A. Offutt. The coupling effect: fact or fiction. In *ACM SIGSOFT Software Engineering Notes*, volume 14, pages 131–140. ACM, 1989.

[14] Kurt Jensen and Lars Michael Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer, 2009.

[15] Wolfgang Reisig. *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.

[16] Michael Westergaard. Cpn tools 4: Multi-formalism and extensibility. In *PETRI NETS 2013*, volume 7927 of *Lecture Notes in Computer Science*, pages 400–409. Springer, 2013.

[17] Stefan Rieger. Etcs specification findings. `https://github.com/openETCS/validation/tree/master/VnVUserStories/ModelVerificationTWT/05-Work/SpecificationFindings`, January 2014.

[18] Bernard Berthomieu and François Vernadat. Time petri nets analysis with tina. In *QEST 2006*, pages 123–124. IEEE Computer Society, 2006.

[19] `https://github.com/openETCS/ERTMSFormalSpecs/tree/master/ErtmsFormalSpecs/doc`.

[20] Michaela Huhn and Stefan Milius. Observations on formal safety analysis in practice. *Sci. Comput. Program.*, 80:150–168, 2014.

[21] Ilyas Daskaya, Michaela Huhn, and Stefan Milius. Formal safety analysis in industrial practice. In *FMICS 2011*, volume 6959 of *Lecture Notes in Computer Science*, pages 68–84. Springer, 2011.

[22] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *QEST 2006*, pages 125–126. IEEE Computer Society, 2006.

[23] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *CAV 2002*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002.

[24] Karsten Wolf. Generating petri net state spaces. In *ICATPN 2007*, volume 4546 of *Lecture Notes in Computer Science*, pages 29–42. Springer, 2007.

[25] Cécile braunstein. Radio communication management -srs subset-026-3.5- sysml model. report, openETCS, 2013.

[26] UNISIG. SUBSET-026 – system requirements specification. SRS 3.3.0, ERA, March 2012.

[27] UNISIG. *SUBSET-026 – System Requirements Specification*, SRS 3.5 Management of the Radio Communictation. In [26], March 2012.

[28] D4.1 openETCS validation & verification plan. Delivrable OETCS/WP4/D4.1V00.09, openETCS, September 2013.

[29] Jörg Brauer, Jan Peleska, and Uwe Schulze. Efficient and trustworthy tool qualification for model-based testing tools. In Brian Nielsen and Carsten Weise, editors, *Testing Software and Systems*, volume 7641 of *Lecture Notes in Computer Science*, pages 8–23. Springer Berlin Heidelberg, 2012.