Proc. of the Intl. Conf. on Advances In Information Processing And Communication Technology - IPCT 2014.

ISBN: 978-1-63248-021-7 doi: 10.15224/ 978-1-63248-021-7-72

On Modeling and Testing Components of the European Train Control System

César Andrés, Ana Cavalli, Nina Yevtushenko, João Santos, and Rui Abreu

***Abstract*—This paper studies the abilities of the formal model of a Timed Extended Finite State Machine (TEFSM) to represent the safety properties of the European Train Control System (ETCS). The model is based on Finite State Machines augmented with continuous variables and time information, which allows representing the basic functioning of the units in this real-time system. In order to represent temporal requirements, timeouts are used for modeling some aspects of the (internal) critical behavior of the train control system. The model abilities to represent safety properties are evaluated using different testing scenarios for model implementations in IF, XML and JAVA languages. Tests are automatically generated using the tool TestGen-IF where corresponding safety properties are specified as test objectives. Based on the obtained experimental results the advantages and disadvantages of a developed model are briefly discussed.**

***Keywords*—European Train Control System, formal model, Timed Extended FSM, active testing, safety properties.**

I. Introduction

European railways have been evolving over the last 150 years within national boundaries, resulting in a variety of different train control systems. To increase the interoperability, the European Union has decided to standardize the European Control Train System, in short ETCS. The standardization has been addressed by developing a system specification which will become mandatory within the European Union in 2015. Several research initiatives attempt to develop frameworks for providing interoperability between the different European train systems [1-5].

The ETCS requirements specification [6] describes the system behavior as well as a number of functional requirements.

César Andrés, Ana Cavalli , João Santos
Telecom SudParis - CNRS SAMOVAR
France

Nina Yevtushenko
Tomsk State University
Russia

Rui Abreu
University of Porto
Portugal

As the significance and complexity of these requirements grow rapidly, formal techniques for producing reliable control software become of utmost importance. Such formal methods and model-based testing are amongst the most promising approaches for increasing software confidence (see, e.g., [7-9]). However, since the functional requirements of the specification of the ETCS[1] are written as plain text, there is a strong need to map these requirements into a formal representation. To formally describe these requirements, one needs a formalism that takes into account different behavior scenarios under different conditions, continuous variables that are related to the train position, speed and acceleration and also different roles of different actors in the specifications: the Radio Block Center (*RBC*), the train (*TRAIN*), and the environment itself. The devised formal model ought to have abilities to represent critical situations such as: a) alarm signals from the *RBC*; b) external inputs to *RBC* and trains; c) critical distance (between two trains or with respect to other obstacles); d) the loss of some messages from/to a train or from/to the *RBC*. In this paper, we propose to use finite state machines augmented with continuous variables and (time) guards to represent the most important requirements of the ETCS and study the abilities of such model for verifying functional and safety properties.

Last years, the use of formal methods for software testing became a reality [see, for example, 10-12], and in the context of this work, active testing scenarios are generated from the model itself; the latter allows to achieve a high degree of automation and certainty. These testing scenarios compile relevant safety properties of the system. To support the theoretical framework a developed model has been implemented in different languages such as XML, JAVA and IF. The TestGen IF tool [12, 13] is utilized to codify the specification and to automatically obtain a set of tests using the Hit-or-Jump Algorithm [13]. The model is first verified by producing an expected output to each test case according to the plain description [6] and then we evaluate the expressiveness of derived tests using mutation testing of a JAVA implementation. The obtained experimental results allow to evaluate the advantages of using a proposed model as well as to identify its limitations.

To summarize, in this paper, we provide a formal model for the requirements of the European Train Control System using Finite State Machines augmented with continuous variables and time constraints. The model is a close representation of the units specification provided by the

[1] The open ETCS project, funded by the ITEA2 program, aims to develop an integrated modeling, development, validation and testing framework for leveraging the cost-efficient and reliable implementation of ETCS. For further information, visit the project website at http://www.itea2.org/project/index/view/?project=10135

Proc. of the Intl. Conf. on Advances In Information Processing And Communication Technology - IPCT 2014.

ISBN: 978-1-63248-021-7 doi: 10.15224/ 978-1-63248-021-7-72

standard. Testing scenarios are automatically generated using the TestGen IF tool. The fault coverage of generated testing scenarios has been evaluated using model implementations in different languages, namely, XML, JAVA and IF and according to the obtained experimental results, the proposed formal model has good abilities for generating high quality tests for detecting faults which threaten the system safety.

The rest of the paper is organized as follows. Section II introduces the basic concepts used for the modeling and testing techniques and presents the formal model. Section III presents a framework for deriving testing scenarios for verifying safety properties of components of the ETCS system. Section IV concludes the paper; in this section, the advantages and disadvantages of a developed model are briefly discussed. We also mention that the preliminary version of this paper has been published as a technical report of Telecom SudParis [14].

II. ETCS Model

In this section, we introduce the basic concepts of the formal model and discuss the decisions we take when building the model and detail the proposed model.

A. *TEFSM model*

A *TEFSM* is an ordinary finite state machine (FSM) augmented with context variables, input/output parameters, predicates and update functions. Given input and output alphabets I and O, we denote by R the set of all *input parameters* and by Q the set of all *output parameters*. The finite set of *context variables* is denoted by V.

A *Timed Extended Finite State Machine*, in short *TEFSM*, is a tuple $E = \langle S, s_0, I, O, T, \Delta, v_0 \rangle$ where S is a non-empty finite set of *states* with the initial state s_0; I and O are *input* and *output* alphabets; T is the set of *transitions*; $\Delta: S \to S \times (N \cup \infty)$, $N = 1, 2, \dots$, is a *timeout function*.

Given the current state s of the TEFSM E, if no input is applied before the timeout $\Delta(s)_{\downarrow N}$ expires, then the TEFSM can spontaneously move to another state $\Delta(s)_{\downarrow S}$ as specified by the timeout function. The special case $\Delta(s) = (s, \infty)$ means that the TEFSM could stay in state s until an input is applied.

A *transition* is a tuple $(s, i, o, Pr, f_{context}, f_{output}, s')$ where $s, s' \in S$ are the *initial* and *final* states of the transition; $i \in I$ and $o \in O$ are input and output actions respectively; $Pr: R \times V \to \{0, 1\}$ is a *predicate* over the input parameters and context variables; $f_{context}: R \times V \to V$ is the *update* function for the context variables; and $f_{output}: R \times V \to Q$ is the *update* function for the output parameters.

B. *Modeling Decisions*

There are many models related to ETCS [see, e.g., 1-5]. Most models describe the system behavior using logic formulas and then verify whether these formulas satisfy some safety requirements, such as different notions of the safe distance, alarm messages (fire, accidents, etc.) which can come from outside the train and the *RBC* as well as from inside the train. As a complementary approach for such verification, testing is commonly used. If the corresponding formula respects the checked safety requirements and an IUT produces only expected outputs to applied test cases then, to some extent, there is a confidence that the model and implementation are safe. In order to develop a formal model in the ETCS context it is necessary to consider the system components and discuss the behavioral aspects of an *RBC* and a train under control and which safety aspects should be taken into account. In this paper, we consider that there are three components, a train under control, the RBC and the environment.

A big portion of safety issues is related for situations when a train under control moves autonomously: when the train should negotiate with the *RBC* about the safety distance, when the train should be stopped, i.e., we have to check the functional aspects of the system. Some core points can be defined where a component has different behaviors, and those points can be considered as states in the model. The conditions when a component moves from one state to another are usually related to different kinds of safety distances, respecting alarm messages, etc. Moreover, transitions significantly depend on the values of continuous variables such as train position, speed, acceleration, etc.

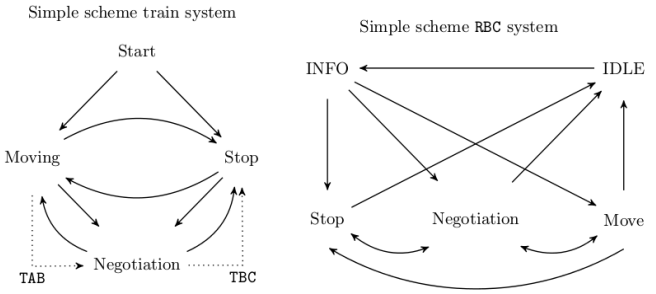

Figure 1. States of the train and RBC models.

For evaluating whether the above safety issues can be solved using the formalism of finite transition systems we consider a simplified TEFSM model. According to our model, there are three components: a train under control, the *RBC* that also knows the position, speed, acceleration of the previous train, and the environment.

A train under control has four special positions where it has different behavior (Figure 1). 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 the *RBC* (for example, from the last linked balise group) and moves to another state (moving/stopping). At this state, the train can get different messages from the *RBC* but almost any message should contain the safety distance according to the position of the previous train or possibly, according to some other obstacles. If the train is in a *safe* position it continues autonomous moving. However, if its position is closer to some *dangerous* point then the train should come to another state and start the *negotiation* with the *RBC*. If the train crosses the dangerous point then the train should be immediately stopped. The safety

Proc. of the Intl. Conf. on Advances In Information Processing And Communication Technology - IPCT 2014.

ISBN: 978-1-63248-021-7 doi: 10.15224/ 978-1-63248-021-7-72

point can be calculated by the RBC based on the data got from the previous train or on some data got from the environment, etc. The train waits for an input from the RBC in order to start moving again.

In our model, we have such a component 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 to the *RBC* if something happens outside. We also model the train "red button" (accident, fire etc. inside the train) 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. The *RBC* also checks whether the train respects the *RBC* instructions. For example, if the train under control sends several consecutive messages where the recommended speed is exceeded more than it is allowed, for the safety reasons, the *RBC* has to stop the train.

Tests verifying the safety aspects should check the transitions in the model, i.e., it should be checked whether the logic expressions for firing a transition are correctly implemented, whether the implementation respects timeouts which model the message loss, etc. Here we notice that there are methods for deriving test cases from a TEFSM [15, 16] which can verify the above aspects, i.e., tests for verifying safety issues of an IUT can be automatically derived based on our model. Moreover, actually, it is known that only FSM models where each input is followed by a corresponding output allow 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 models to some kind of an FSM (see, e.g., [17]). Below we describe the train and the *RBC* TEFSMs in more details.

C. *Modeling requirements*

In this subsection, some requirements of the ETCS Level 3 are presented. At 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 the permission.

According to the set of requirements, for each train, there is the safety distance *d*. In our simplified model, the *critical point* of $Train_j$ of interest is calculated by the *RBC* that controls the $Train_j$ in order 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.

If $Train_j$ is controlled by the *RBC*, then the train reports its current position (*p*), speed (*v*), and acceleration (*a*) and the current internal state; the output parameters *p*, *v*, *a* are updated according to the information about the train.

The *RBC* analyzes the available information, in particular, obtained from the previous train, and returns the critical distance *d* to the train. According to the rules, we describe the safety distance requirements that are used in the model. For the sake of simplicity, in our model, the input parameter *SD* represents the safety distance and is a constant in the model. We say that a train at position *p* and with the critical point *d* is: a) in a *safe* position if $(d - p) > 4 \cdot SD$; b) in a *negotiation* position if $2 \cdot SD \leq (d - p) \leq 4 \cdot SD$; c) in a *stop* position if $(d - p) < 2 \cdot SD$. If the train is at the *Moving* state and it is in a *safe* position then the train can remain at this state having the speed and acceleration under limits 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 enters the *Negotiation* state. Finally, if the train is at the *Negotiation* state and the critical distance does not increase then the train enters the *Stop* state.

The model presented in this paper also satisfies the following requirement: "*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*". In order to model this situation, our model is augmented with the timeouts TAB and TBC.

D. *Formal Model*

1) *The train*. A *TEFSM* that describes the behavior of the train 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 and if the train crosses the *Negotiation* point then the train and the *RBC* start negotiating and the train moves to the *Negotiation* state.

If the train reaches the *Stop* position then the train should be stopped, i.e., the train moves to the *Stop* state. 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*.

There is a timeout TAB (the dot line) from the state *Moving* to the *Negotiation* state. 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 the *Negotiation* state. There also is a timeout TBC from the *Negotiation* state to the *Stop* state. If the train is in the *Negotiation* state and the train does not receive any input during TBC time units, then the train automatically moves to the *Stop* state.

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

Proc. of the Intl. Conf. on Advances In Information Processing And Communication Technology - IPCT 2014.

ISBN: 978-1-63248-021-7 doi: 10.15224/ 978-1-63248-021-7-72

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}) sent to the train contains the critical point *d* and the maximum speed v_{max} that the train can have.

Depending on the position of the previous train or for some other reasons, 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. 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 yet, i.e., the train is in a *negotiation* position. The *Move* state denotes that the train is autonomously moving. The train can be stopped when 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. In order to stop the train, the *RBC* sends the message *stopRBC*. If the train receives this message at any state then the train moves to the *Stop* state. The message *neg*(*d*, v_{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 the *RBC* sends the message *stopRBC* in order to stop the train and enters the *Stop* state.

III. Test derivation

In this paper, we assume that the *RBC* and the train are tested separately, i.e., when testing a train implementation the *RBC* is replaced by a tester that sends inputs to the train and checks whether the produced outputs are expected.

A. *Test derivation by TestGen-IF tool*

A set of tests which check appropriate safety properties was automatically derived using the train description in the *IF language* where basic safety requirements can be described as appropriate test purposes (test objectives). Based on these test objectives, the toolset TestGen-IF [11, 12] automatically generates a set of tests based on the Hit-or-Jump test generation algorithm [13]. After defining a test objective, the tool provides a test suite associated with a corresponding scenario. In our case, we have generated tests for 20 test objectives. Each test has been checked for meeting the system requirements specification [6]. The fault coverage and the expressiveness of the generated test suite has been evaluated using other representations of the train model, namely, XML and JAVA representations.

In order to be confident that the set of test objectives considered for test generation has a good coverage over necessary safety properties of the train system, we evaluate the quality of IF tests with respect to safety properties of the train system. For this purpose, an XML train simulator has been implemented. At the next step, XML mutants of different types which threaten safety issues of the train system, were created from which a corresponding JAVA code was automatically generated. The fault coverage of tests generated by the TesGen-IF tool was evaluated over generated JAVA mutants.

To verify if the test suite with 20 test cases generated by the TestGen-IF tool can detect faults which threaten the system safety, we ran the above test suite against JAVA implementations generated from different XML mutants which have corresponding injected faults that violate the safety properties of the train system. We inserted faults of the following types: i) incorrectly implemented a destination state of a transition; ii) the implementation does not correctly identify the received inputs; iii) an output of a transition is wrong; iv) the conditions under which a transition can be fired are wrong; v) update function for internal variables and/or output parameters are wrongly implemented; vi) an implementation does not respect the specification time constraints. All types of faults have been successfully detected by multiple test cases and each test case was able to detect faults of different types. In other words, a derived test suite has a good quality, since it detects different types of faults that affect the safety issues. We also notice that the studied test suite does not have considerable redundancy, since different faults are detected by different test cases.

It should be noticed that there are other test generation tools [see, for example, 18] which can also be used for the considered model; nevertheless, we used the TestGen-IF tool, since IF has a model checker that can be later used for the model verification.

IV. Concluding remarks

This paper proposes a formal model that is a finite state machine augmented with continuous variables and time constraints (TEFSM) to represent requirements of the European Train Control System. The abilities of the model for representing safety properties and for deriving tests for checking these properties have been evaluated. Tests are automatically derived using the IF representation of the proposed model and the toolset TestGen-IF. Based on performed experiments the conclusion can be drawn that the TEFSM formalism is adequate for modeling and testing safety ECTS properties. Nevertheless, the experiments also eliminated some limitations of the considered model. First, output timeouts which represent a transition duration have to be taken into account. Secondly, more complex update formulae should be considered and some model states have to be composite, i.e., the model has to be hierarchical. As a result, it seems to be better to use UML-based languages when describing the behavior of ETCS units at higher levels.

References

[1] M. Hörste and E. Schnieder, "Modelling and simulation of train control systems using petri nets," World Congress on Formal Methods in the Development of Computing Systems, FM'99, vol. 3. 1999, pp. 1-16.

ISBN: 978-1-63248-021-7 doi: 10.15224/ 978-1-63248-021-7-72

[2] 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., vol. 5, no. 1, pp. 3–21, Jan. 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=1241714. 1241716 =0pt

[3] A. Zimmermann and G. Hommel, "Towards modeling and evaluation of etcs real-time communication and operation," J. Syst. Softw., vol. 77, no. 1, pp. 47–54, Jul. 2005.

[4] G. Bundell, "Aspects of the safety analysis of an on-board automatic train operation supervisor," in IEEE Int. Conference on Systems, Man and Cybernetics, 2009. SMC'09. IEEE, 2009, pp. 3223–3230.

[5] A. Platzer and J. 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. Springer, 2009, pp. 246–265.

[6] European Research Agency, "ERTMS/ETCS, functional requirements specification," European Union, http://www.era.europa.eu/Document-Register/Pages/ERA-ERTMS-03204.aspx, Brussels, Belgium, 2007.

[7] J. Faber, "Verification architectures: Compositional reasoning for real-time systems," in 8th Int. conf. on Integrated formal methods, IFM'10. Springer, 2010, pp. 136–151.

[8] A. Post, J. Hoenicke, and A. Podelski, "rt-inconsistency: a new property for real-time requirements," in Fundamental Approaches to Software Engineering. Springer, 2011, pp. 34–49.

[9] A. Post and J. Hoenicke, "Formalization and analysis of real-time requirements: a feasibility study at bosch," in Verified Software: Theories, Tools, Experiments. Springer, 2012, pp. 225–240.

[10] J. Peleska, "Industrial-strength model-based testing - state of the art and current challenges," in 8th Workshop on Model-Based Testing, MBT'13, 2013, pp. 3–28.

[11] A. Cavalli, E. M. D. Oca, W. Mallouli, and M. Lallali, "Two complementary tools for the formal testing of distributed systems with time constraints," in 16. 12-th IEEE/ACM International Symposium on Distributed Simulation and Real Time, DS-RT'08. IEEE Press, 2008, pp. 315–318.

[12] I. Hwang, A. Cavalli, M. Lallali, and D. Verchere, "Applying formal methods to PCEP: an industrial case study from modeling to test generation," Software Testing, Verification and Reliability, vol. 22, no. 5, pp. 343–361, 2012.

[13] A. Cavalli, D. Lee, C. Rinderknecht, and F. ZaÃ¯di, "Hit-or-Jump: An algorithm for embedded testing with applications to in services," in Formal Methods for Protocol Engineering and Distributed Systems, ser. IFIP Advances in Information and Communication Technology. Springer, 1999, vol. 28, pp. 41–56.

[14] C. Andres, A.R. Cavalli, N. Yevtushenko. On modelling and testing the european train syste. Tech. report TechRca 14-03-2013. Telecom SudParis, 2013.

[15] M. Zhigulin, N. Yevtushenko, S. Maag, and A. Cavalli, "FSM-based test derivation strategies for systems with time-outs," in 11th Int. Conf. on Quality Software, QSIC'11, 2011, pp. 141–149.

[16] M. G. Merayo, M. Núñez, and I. Rodrguez, "Formal testing from timed finite state machines," Computer Networks, vol. 52, no. 2, pp. 432–460, 2008.

[17] J. Springintveld, F. Vaandrager, and P. R. D'Argenio, "Testing timed automata," Theoretical computer science, vol. 254, no. 1, pp. 225–257, 2001.

[18] A. Belinfante, "JTorX: A tool for on-line model-driven test derivation and execution," in 16th Workshop on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'10, LNCS 6015. Springer, 2010, pp. 266–270. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12002-2_21

About Author (s):

César Andrés has received his Ph.D. degree from the Madrid University. IN 2012-2013 he was a postdoc at Télécom SudParis in France with his research focusing on the active and passive testing of communicating systems.

Ana Rosa Cavalli is Full Professor at TELECOM SudParis since 1990, and nowadays she is the director of the Software for Networks department. Her research interests are on formal modeling, testing methodologies for conformance and interoperability testing, active testing and monitoring techniques, validation of security properties and their application to services and protocols

Nina joined Tomsk State University шт 1991 as a professor and presently she leads a research team working on the synthesis and analysis of discrete event systems. Her research interests include formal methods, automata theory, distributed systems, protocol and software testing.

João Santos is a second year PhD student in Télécom SudParis in France with his research focusing on the diagnosis of time constrainted systems. He completed his master thesis in the Faculty of Engineering of the University of Porto in 2007.

Rui Maranhão (publishes as **Rui Abreu**) graduated in Systems and Computer received his Ph.D. degree from the Delft University of Technology, the Netherlands, in November 2009, and he is currently an assistant professor at the Faculty of Engineering of University of Porto. He is also with the School of Computer Science of Carnegie Mellon University (CMU), USA, as a Visiting Faculty Member.