

Verifying Real-Time Aspects of the European Train Control System*

Johannes Faber

Department of Computing Science, University of Oldenburg, Germany
johannes.faber@informatik.uni-oldenburg.de

1 Introduction

In this real-time case study, we formally specify a part of the *European Train Control System* (ETCS) [ERT02, ECS99] with the specification language CSP-OZ-DC treating the handling of emergency messages. We apply the methods from [HM05] to enable the formal verification of the model. In addition, we pick up the idea of using *Fault Tree Analysis* as a decomposition technique [Sch03] to simplify the verification tasks of this profoundly parallel case study model and adapt it to the usage with CSP-OZ-DC.

We present the first real-world application of CSP-OZ-DC and give a realistic, object-oriented, and holistic model of a ETCS subsystem, considering the communication, processing, and real-time behaviour. We explicitly take infinite data types and messages with potential infinite parameters into account. Other work on ETCS case studies like [ZH05, HJU05] focus on the stochastic examination of the communication reliability and model components like the train and the RBC in a rather abstract way without considering the internal behaviour.

2 ETCS Case Study: Emergency Message Handling

The emerging ETCS is an international train control system by the European Commission that shall replace traditional, national train control systems in the future to ensure cross-border interoperability and improve railway safety and track utilisation. In the final ETCS implementation-level 3 [ERT02, ECS99], the currently existing national trackside systems for detection of train speed, location, and integrity are not used anymore. Instead current parameters for a moving train are ascertained in cooperation of the Train's on-board ETCS unit with an appropriate *radio block centre* (RBC), which controls the traffic in a well-defined area and grants *movement authorities* (MA) to trains. RBC and trains communicate over a GSM-R radio connection. A main issue

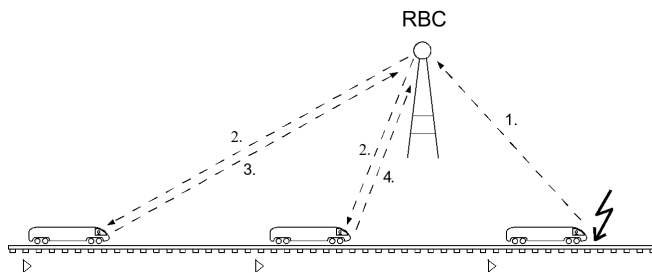


Figure 1: Consecutive trains.

of ETCS is increasing the possible traffic density. Therefore the *moving block principle* is used, by which the MA's are always given up to a position known as the *safe rear end* of the preceding train. This allows trains to minimise the gap between them.

In our case study, we consider the treatment of emergency messages as required for ETCS level 3. The situation may be sketched as follows:

If consecutive trains are driving with a minimal distance according to the moving block principle, the train control system has to guarantee that in case of an accident of the first train the other trains come to a standstill before reaching the preceding train.

For achieving this behaviour securely, the following procedure shall be applied, cf. Fig. 1. When the first train detects an emergency situation it has to send an *emergency message* to the appropriate RBC. The maximum end-to-end delay for a message transfer is 500 msec. After receiving the emergency message the RBC must, in less than 0.5 seconds, forward the emergency message to every train approaching the danger position. The trains have to acknowledge these messages. If necessary to avoid a collision, the emergency brakes of these trains are automatically applied after less than 0.5 seconds. If not necessary, the driver shall maintain control on the train. Hence, a (visual and audible) indication shall be given to the driver of the train after less than 1.5 seconds. He has up to 5 seconds to acknowledge the receipt of the message. After this time the emergency brakes of the train have to be applied automatically.

*This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See <http://www.avacs.org> for more information.

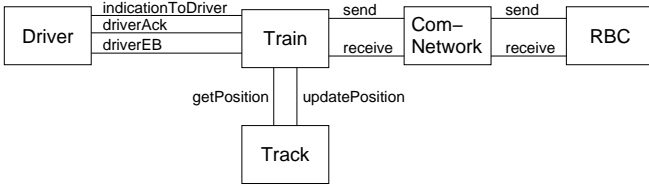


Figure 2: Components of the case study.

Specification Complex systems like the ETCS are often determined by several components running in parallel, by the communications between these components, by internal data and state changes, and by real-time aspects. Hence, [HO02, HM05] introduce *CSP-OZ-DC*, which is a combination of three well-known specification languages:

- The external and internal *communications* of the parallel components are described with Communicating Sequential Processes (CSP) [Hoa85]. For instance, the main-process of a train comprises the interleaving of three subprocesses:

$\text{main} \stackrel{c}{\equiv}$

$\text{Running} \parallel \text{HandleEM} \parallel \text{EmergencyDetection}.$

The processes of several components communicate over *channels* that enable the transfer of parameters. For example, the trains and the RBC communicate over the channels *send* and *receive*. The RBC sends an emergency warning to a train with its subprocess

$\text{WarningTo}(id : \text{TrainID}) \stackrel{c}{\equiv}$

$\text{send.} \text{EmergencyWarning!}id \rightarrow \dots$

whereas trains receive this message with following subprocess:

$\text{HandleEM} \stackrel{c}{\equiv} \text{receive.} \text{EmergencyWarning.ID} \rightarrow \dots$

- *Data aspects* are specified with Object-Z (OZ) [Smi00]. Therefore, CSP-OZ-DC is an object-oriented specification language and we can model ETCS components like trains, RBC's, and also drivers in a natural way as classes. Fig. 2 depicts the components of this case study and their connecting channels.
- *Real-time constraints* are described using the real-time logic Duration Calculus (DC) [ZH04]. Since the full DC is too powerful for automatic verification, we only use *counterexample formulae* [HM05], which specify undesired behaviour using a timed trace. For instance, the following trace states that it is forbidden that after an *indicationToDriver*-event no acknowledgement (*driverAck*) follows for more than five time units:

$\uparrow \text{indicationToDriver} ; \exists \text{driverAck} \wedge 5 < \ell.$

In [HM05] a semantics for CSP-OZ-DC is given by Phase-Event-Automata (PEA), a new class of timed-automata that synchronises on both events and states. PEA's handle variables in a non-deterministic way, i.e. if no explicit value update exists on a transition, the

new value will be chosen non-deterministically. The semantics is compositional in the sense that every part (CSP/OZ/DC) of every component is translated to a single PEA; the whole specification comprises the parallel product of all these automata. This enables the compositional verification, since for checking real-time properties it often suffices to consider only such automata that specify real-time properties (while the CSP- and OZ-parts can safely be neglected).

3 Verification

There are several safety properties we want to verify in this case study. Certainly, one goal is to show that two consecutive trains will never collide. Though we will restrict ourselves to the simpler property “The time between an emergency alert and the proper reaction of the train will never exceed 7.5 seconds”. We verify the safety property “Reaction time not exceeded” using the constraint-based abstraction-refinement model-checker ARMC [Ryb02]. ARMC is well-suited for the verification of PEA's because it can deal with infinite data types and non-deterministic variable assertions. In simple cases, where we could abstract from complicated variable expressions, we also succeeded with the real-time model-checker Uppaal [UUP05], which checks our examples very fast (less than a second). But in general, PEA's cannot directly be transformed into a Uppaal-model because of the non-deterministic assertions in CSP-OZ-DC.

Fault Tree Analysis Unfortunately, due to the state explosion problem it is not possible to check the entire model in a single step. To solve this problem, we apply *Fault Tree Analysis* (FTA) as a decomposition technique that allows us to verify the desired property step by step.

Originally, FTA is a standardised technique [IEC90] used by engineers to explore possible failure states of safety-critical systems. The aim of FTA is to analyse conditions and factors causing an undesired, so-called *top event*. The fault tree itself is a graphical representation of these causes as a tree where the top event is stated as the root node. The other nodes represent either events that are direct or indirect causes of the top event or gates that join two or more events. [STR02] presents a formal fault tree semantics and [Sch03] proposes to use fault trees as a decomposition technique to simplify verification tasks. We adapt this approach to PEA's and analyse the case study with FTA.

Figure 3 depicts a fault tree of our case study. The top event E1 we want to analyse is that the reaction-time of the train exceeds 7.5 seconds. Obviously (cf. Fig. 1), this fault occurs either on the side of the RBC and the message transfer (E2) or on the side of the train in the message processing (E3). In other words, too much time elapses between the detection of the failure and the notification of the following train or after receiving the emergency warning the train does not apply the emergency brakes and inform the driver. Thus, we can decompose the top event E1 with the events E2 and E3. In Fig. 3 these fault events are connected by an or-gate

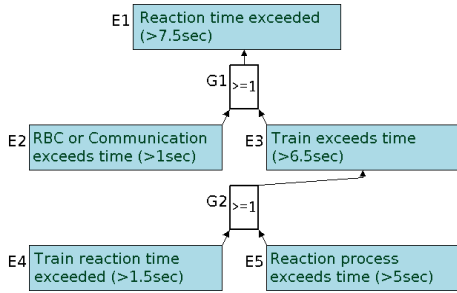


Figure 3: A fault tree for the ETCS case study.

(G1), i.e. the occurrence of one of the cause events suffices to provoke the consequent top event. The second gate (G2) is a decomposition of the fault-event E3 into two smaller fault-events E4 and E5.

The benefit of this procedure is that we do not have to verify the safety property at once. Instead we perform two verification tasks:

1. We show that events at the leaves of the tree will never occur.
2. We show for every gate that the consequent fault only occurs if one of the cause events occur.¹

These two properties ensure that the main-fault will never occur.

Verification-Task 1. We formulate the fault events of the tree in a sublogic of the DC. [Mey05] defines a subclass of the DC, so-called traces, that can be transformed into PEA’s and that also have automata for their negations. Such *test-automata* enter a “bad” state if the corresponding DC-formula, representing undesired behaviour, is fulfilled. The test-automata run in parallel with the PEA’s influencing the property that is to be checked. For instance, if we want to check whether the fault event E2 can ever occur, we have to select two PEA’s that represent the real-time behaviour of the train and the communication layer. Then we build the parallel product of these automata and the test-automata for the event E2. In addition, we have to transform this parallel product in a constraint representation that can be read by ARMC. Finally, we can verify that the “bad” states of the test-automaton are not reachable.

Verification-Task 2. To show the second verification task for a given gate, we generate PEA’s for the negated cause events, i.e. we temporarily assume that these events will not occur, and a test-automata for the consequence event. Then we build the parallel product, if necessary with some automata of the model, and check the reachability of the “bad” states of the test-automata. If they are not reachable we have shown that under the assumption that the causes of a gate do not occur, the consequence event does also not occur. That is, the second property is fulfilled for the current gate.

In the ETCS-example we have overall five verification tasks to do: one verification run for both of the gates and three verification runs for the leaves.

¹In terms of [STR02] such a property is called *complete*.

Task	Prod.(1)	Tran.(2)	Veri.(3)	Total
Event E2	3.031s	3.692s	1.357s	8,080s
Event E4	1.420s	0.432s	0.272s	2,124s
Event E5	1.083s	0.177s	0.048s	1,308s
Gate G1	1.833s	1.251s	0.614s	3,698s
Gate G2	1.831s	1.291s	0.550s	3,672s
Total	9.197s	6.843s	2.813s	18,853s

Table 1: Experimental results.

4 Results and Future Work

Up to now, we have verified that the entire emergency message procedure is finished in time, i.e. the train applies the emergency brakes or the driver takes over the responsibility for driving. Table 1 lists the experimental results for every verification step: Building the parallel product (with first simplifications of the formulae) and adding automatically generated test-automata (1), transforming the product into a constraint-based representation (2), and model-checking with ARMC (3). The fault tree guided analysis, that is applied the first time to CSP-OZ-DC and PEA’s here, leads in an intuitive way to a decomposition. In our case we get five simple verification steps that can be checked very fast. Currently we are working on a decomposition of the more comprehensive safety property “The trains do not collide.”

We have tool support [PEA05] for the verification steps mentioned in Tab. 1 so far. Secondly, there exists graphical editors [Mob05] for PEA’s (Moby/PEA) and fault trees (Moby/FT). We intend to implement a parser for CSP-OZ-DC and support for fault tree handling with PEA’s to complete the verification process.

Our work is the first approach to model and verify relevant parts of the ETCS with the new specification language CSP-OZ-DC, which allows us to consider the system in a holistic way, including the communication and timing behaviour as well as the component’s internal processings. The language is object-oriented and therefore we think it is well-suited for larger models. This is the first step towards our vision to realise comprehensive parts of the ETCS using CSP-OZ-DC. In the context of AVACS [AVA05] we also work on automatic code generation from CSP-OZ-DC, which will directly ensure, in combination with our verification approaches, the correctness of the implementation.

References

- [AVA05] The AVACS project page. Transregional Collaborative Research Center 14 AVACS, <http://www.avacs.org>, 2004-2005.
- [ECS99] ECSAG. ERTMS/ETCS Functional requirements specification. <http://www.aEIF.org/ccm/default.asp>, 1999. Version 4.29.
- [ERT02] ERTMS User Group, UNISIG. ERTMS/ETCS System requirements specification. <http://www.aEIF.org/ccm/default.asp>, 2002. Version 2.2.2.

- [HJU05] H. Hermanns, D.N. Jansen, and Y.S. Usenko. A comparative reliability analysis of ETCS train radio communications. Technical Report 2, SFB/TR 14 AVACS, 2005. <http://www.avacs.org>.
- [HM05] J. Hoenicke and P. Maier. Model-checking of specifications integrating processes, data and time. In J.S. Fitzgerald, I.J. Hayes, and A. Tarlecki, editors, *FM 2005*, volume 3582 of *LNCS*. Springer-Verlag, 2005.
- [HO02] J. Hoenicke and E.-R. Olderog. CSP-OZ-DC: A combination of specification techniques for processes, data and time. *NJC*, 9, 2002.
- [Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- [IEC90] IEC 61025: Fault tree analysis (FTA), 1990.
- [Mey05] R. Meyer. Model-Checking von Phasen-Event-Automaten bezüglich Duration Calculus Formeln mittels Testautomaten. Master's thesis, University of Oldenburg, 2005. In German.
- [Mob05] Moby product home page. Department of computing science, University of Oldenburg, <http://csd.informatik.uni-oldenburg.de/~moby>, 1998-2005.
- [PEA05] PEA-Toolbox. Department of Computing Science, University of Oldenburg, http://csd.informatik.uni-oldenburg.de/~jfaber/avacs_en.html, 2005.
- [Ryb02] A. Rybalchenko. A model checker based on abstraction refinement. Master's thesis, Universität des Saarlandes, 2002.
- [Sch03] A. Schäfer. Combining real-time model-checking and fault tree analysis. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME '03*, volume 2805 of *LNCS*. Springer-Verlag, 2003.
- [Smi00] G. Smith. *The Object Z Specification Language*. Kluwer Academic Publishers, 2000.
- [STR02] G. Schellhorn, A. Thums, and W. Reif. Formal fault tree semantics. In *IDPT '02*, 2002.
- [UUP05] Uppaal home page. Basic Research in Computer Science (University of Aalborg) and Department of Computer Systems (University of Uppsala), <http://www.uppaal.com>, 1995-2005.
- [ZH04] Zhou Chaochen and M. R. Hansen. *Duration Calculus*. Springer-Verlag, 2004.
- [ZH05] A. Zimmermann and G. Hommel. Towards modeling and evaluation of ETCS real-time communication and operation. *The Journal of Systems and Software*, 77(1), 2005.