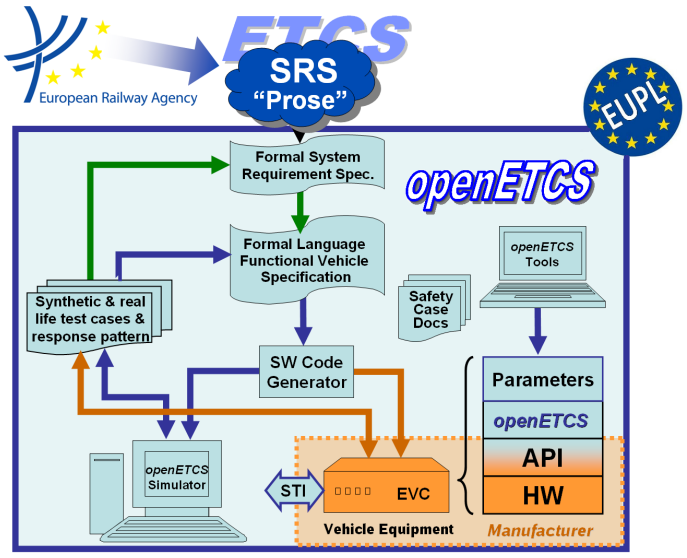
Universität Augsburg
Fakultät für Angewandte Informatik

DB NETZE



Funded by:

Federal Ministry of Education and Research

Région de Bruxelles-Capitale

GOBIERNO DE ESPAÑA MINISTERIO DE INDUSTRIA, ENERGÍA Y TURISMO

Master's thesis

Modeling of a safety-critical system for ensuring the interoperability in the European railway traffic

Alexander Probst

2015-10-02

APPROVED FOR PUBLIC RELEASE.
DISTRIBUTION IS UNLIMITED

Author
Alexander Probst

Reviewer
Prof. Dr. Wolfgang Reif

Second Reviewer
Prof. Dr. Bernhard Bauer

Advisors
Johannes Leupolz, Dipl.-Inf. (University of Augsburg)
Baseliyos Jacob, MBA, Dipl.-Ing. (FH) (DB Netz AG)

Technical assistance
Bernd Hekele, Dipl.-Math. (DB Netz AG)
Peyman Farhangi, Dipl.-Inf. (DB Netz AG)

Abstract

The European Train Control System (ETCS) shall replace in the intermediate and long-term all existing national train protection systems in Europe and ensure interoperability in the European railway traffic. Although 4000 km of the European rail network are already equipped with ETCS, there exists until now no fully functional ETCS on-board unit that is compatible with all ETCS railway lines. This issue shall be tackled with the international research and development project openETCS. Its main objective is the formal methods based development of a vendor-neutral software reference implementation of an ETCS on-board unit that is compatible with all ETCS equipped railway lines. In cooperation with Deutsche Bahn Netz AG as a project leader of openETCS, a Radio Block Center (RBC), which is part of ETCS, was modeled by means of a formal specification. Afterwards, a test of the resulting model was performed in order to find critical errors in the specification.

Acknowledgments

I would like to express my gratitude to my head of department Klaus-Rüdiger Hase who gave me the opportunity to work for the openETCS project. I would like to thank my advisor Baseliyos Jacob who spent a lot of time to teach me the basics of ETCS. Many thanks go also to my colleagues Bernd Hekele and Peyman Farhangi. They helped me to understand the principles of the SCADE language and invested a lot of time in me.

Furthermore, I would like to thank Prof. Dr. Reif and my advisor Johannes Leupolz, who supervised my thesis at the University of Augsburg.

Last but not least, I would like to thank to the many authors who make their articles freely accessible on the web.

Table of Contents

Acronyms

DMI Driver Machine Interface.

EN European Standard.
EoA End of Authority.
ERA European Railway Agency.
ETCS European Train Control System.
EU European Union.
EUPL European Union Public License.
EVC European Vital Computer.

FIFO First In First Out.
FLOSS Free/Libre Open Source Software.

GSM-R Global System for Mobile Communications - Railway.

JRU Juridical Recording Unit.

LoA Limit of Authority.
LRBG Last Relevant Balise Group.

RBC Radio Block Center.

SCADE Safety Critical Application Development Environment.
SIL Safety Integrity Level.
SRS System Requirements Specification.

1 Introduction

1.1 Background

Failures in safety-critical systems claimed in the last centuries many lives and caused environmental damage and high economic losses. The linear accelerator Therac-25 was e.g. used in the eighties in radiotherapy for the treatment of cancer. In the years from 1985 until 1987, three people died due to radiation sickness because the device emitted at certain times a radiation overdose. Three other people were seriously injured. The reason for this disaster was a software bug and a lack of quality assurance in the development process. [57] In the year 1996, a software failure in the Ariane-5 rocket system led to an explosion of the missile after its launch. This incident is with an amount of 370 million US-Dollar still one of the most expensive economic losses that was caused by software. [17]

Nowadays, manually and mechanically controlled systems in trains and railway infrastructures are more and more replaced with modern electronics that are software-intensive, i.e., that software is the dominating component and that most of the functionality is achieved via it rather than with hardware implementations [77]. Train protection systems are part of it. They belong to the category of safety-critical railway signaling systems and are technical installations that monitor, control and influence train runs depending on certain environmental influences. In case of violation of allowed parameters, the system actively intervenes into the route of the train and performs all necessary steps to avoid dangerous situations.

A train protection system that is now in the focus of interest for the European transportation service providers is the European Train Control System (ETCS). ETCS shall replace in the intermediate and long-term all existing national train protection systems in Europe and ensure interoperability in the European railway traffic. Although technical specifications are available, there exists until now no fully functional ETCS on-board unit, also called European Vital Computer (EVC), that is compatible with all ETCS railway lines. Therefore, the international research and development project openETCS was initiated. The aim of this project is the formal methods based development of a vendor-neutral software reference implementation of an EVC that is compatible with all ETCS equipped railway lines.

Railway accidents like in the Swiss Lötschberg Base Tunnel in which an ETCS software error led to a derailment of a freight train show that safety-critical systems demand high safety requirements (see Fig. 1) [75]. These requirements can be only achieved by adhering strict rules in testing and documentation and using certain industrial standards in hardware and software engineering.

1.2 Aims

In this master's thesis and in cooperation with Deutsche Bahn Netz AG as a project leader of openETCS, a Radio Block Center (RBC) shall be modeled by means of a formal specification.

Fig. 1. Derailment of a freight train in the Swiss Lötschberg Base Tunnel due to an ETCS software error [75, p.5]

The RBC is part of ETCS and a trackside system that exchanges track and train related information with an EVC of a train via a GSM-Railway (GSM-R) connection. Since the RBC is a safety-critical system, all safety properties that are prescribed by standards and regulations must be strictly adhered. It must be ensured that the final system is consistent and free from errors. This is accomplished by the strict application of formal methods during the development.

Chapter 2 analyzes the current situation in the European railway traffic and outlines the aims that shall be reached with the openETCS project. Chapter 3 gives an introduction to formal methods and briefly describes the role of formal methods in the development of railway signaling systems. Moreover, the SCADE [18] tool, which is used for a formal methods based development within the openETCS project, is introduced. Chapter 4 explains the functionality of the RBC to provide necessary basic knowledge for the formal specification. In chapter 5, the RBC is formally specified in SCADE. In chapter 6, the resulting model is tested for critical errors. With this structure, the master's thesis is a good entry-point to the field of formal methods in railway signaling systems and enables the reader to get an impression of the state-of-the-art. In combination with the project openETCS and the development of the RBC model as a part of it, the thesis is able to show how formal methods can be applied beneficially to an industrial safety-critical system.

1.3 Application of formal methods in the industry

Formal methods are meanwhile an accepted technique in the industry for the development of computer systems and there is a variety of applications for them [9, p.189]. Transport is an example for it. Thereby, railway signaling systems can be considered as one of the most important areas in which formal methods are applied. According to the review from [7], the amount of successful projects and relevant publications in the past can be taken as an indicator for this. A survey from [88] investigated 62 well known industrial projects from the past that used formal methods in their development process. Figure 2 shows the application domains of these projects. It can be clearly seen that transport is the largest single application domain for formal methods, followed by the financial sector.

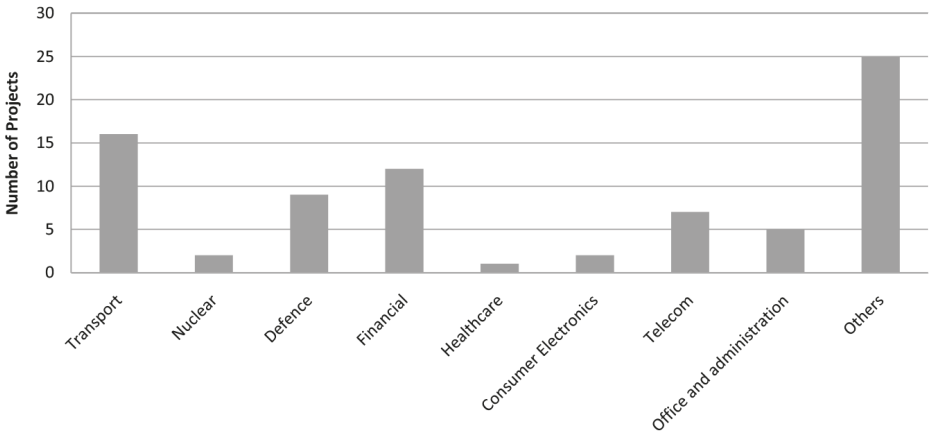


Fig. 2. Application domains for formal methods in the industry [88, p.19:6]

1.4 Related work

Formal methods are today an active research area. Railway related topics are, among others, discussed in the *FM-RAIL-BOK 2013* [13], which contains selected topics about railway control and safety systems. An example how formal methods can be applied in safety-critical railway signaling systems is discussed in [29]. Thereby, hybrid automata [44] and Simulink/ Stateflow charts [60] were used for the modeling of platform screen doors and collision avoidance in subway control systems. Simulink/Stateflow charts were also used in the Metrô Rio [30] case study for designing an automatic train protection system. [58] gives an overview how formal methods can be applied for the requirement analysis of safety-critical train control systems. In this work, requirements were translated

to a formal Property Specification Language (PSL) model [2] and checked with the requirement analysis tool RATSY [8]. Another work on train control systems was proposed by [47]. It used Z notation [51] and statecharts [35,36] for the specification and verification. There are also publications available that try to formalize some parts of the functionality of a RBC. [45] modeled the behavior of the GSM-R communication with the formal process algebra based modeling and specification language MoDeST [14]. [11] formalized scenarios for a RBC in cooperation with Ansaldo STS and used Message Sequence Charts (MSC) [50] and the Structured Description Language (SDL) [49] for the formal specification of the system. Unfortunately, it must be pointed out that none of the existing RBC formalizations is able to deal with all the complex requirements that are prescribed by the ETCS specification.

2 ETCS and the interoperability in the European railway traffic

2.1 Current situation

Wealth and employment are important criteria for a successful economy. Efficient transportation networks can make an important contribution to that. Globalization is fostering a continuously growing flow of goods at international level. It is nowadays for the most people a self-evident fact that a product is available anywhere and at any time. This clearly shows the importance of a cross-border freight traffic.

The European Commission assumes that the freight traffic will increase around more than two-thirds between the years 2000 and 2020 within the European Union. Thereby, the cross-border freight traffic will record the greatest increase between the single member states. [21, p.6]

With 407.2 billion tonne-kilometers in the year 2012, the European rail freight transport takes an important economic role [23, p.34]. However, that does not change the fact that the cross-border railway traffic has serious competitive disadvantages towards another transport systems such as roads. Within the European Union, there exist seven different track gauges, up-to six different traction and more than twenty different train protection systems (see Fig. 3)[23, p.80].

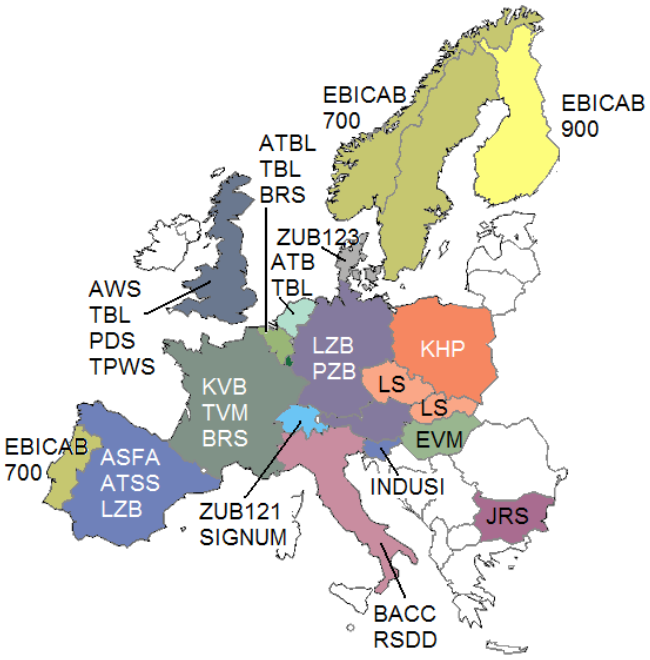


Fig. 3. Train protection systems in Europe [67]

These grievances have historical reasons. Railway traffic was for a long time a national issue. This led, amongst other things, to the development of train protection systems that base on national technical standards. Other relevant aspects also came into play such as the introduction of country-specific and legally established operating procedures, a lack of cooperation between the transportation service providers at European level and a lack of regulatory interest by the European governments. All these factors led on the bottom line to a lack of interoperability in the European railway traffic.

Especially the different train protection systems represent a big hurdle for the transportation service providers. On the one hand, a cross-border railway traffic can be accomplished by an upgrade of the rolling stock to multi-system locomotives. This solution requires the integration of mutually incompatible national train protection systems in the same vehicle. This results, however, in elaborate and costly approval procedures. Another possibility is to change the rolling stock at the frontier stations. In the long-term view, this is also a costly and inflexible solution.

In order to counteract these issues, uniform technical standards are necessary. The European Union (EU) and other organizations support therefore the introduction of the train protection system ETCS. ETCS shall replace in the intermediate and long-term all existing national train protection systems in Europe and ensure an interoperable European railway traffic (see Fig. 4). Furthermore, it shall boost the European economy, eliminate the competitive disadvantages towards another transport systems, reduce the costs for the transportation service providers and increase safety standards. [25]

Fig. 4. Interoperability in the European railway traffic with ETCS [67]

To push the introduction of ETCS forward, the EU dictates with the council directive 96/48/EC [19] that all high-speed lines of the trans-European rail network have to be equipped with ETCS. The council directive 2001/16/EC [20] defines similar requirements for conventional railway lines. Both of them were merged in the meantime into the council directive 2008/57/EC [22].

However, only about 4000 kilometers of track are in Europe equipped with ETCS. Although the European Railway Agency (ERA) created with the system requirements specification (SRS) 3.3.0 [26] a publicly available specification of ETCS, there exists until now no fully functional ETCS on-board unit that is compatible with all ETCS equipped railway lines. One of the main reasons for that is the human factor. On the one hand, the specification documents are created by international railway experts with different know-how and cultural backgrounds. On the other hand, national operating rules are still an important part in ETCS. Moreover, the specification documents are written in prose and have therefore much room for interpretation. The result is a highly complex specification that is nevertheless not precise enough for a direct development. This can lead to inconsistent vendor-specific hardware and software specifications that result in incompatible products between the single manufacturers.

2.2 Project openETCS

The above mentioned issues shall be tackled with the international research and development project openETCS[1,2], which currently consists of twelve German and eighteen European partners [67]. The main objective of the openETCS project is the development of a vendor-neutral software reference implementation of an EVC according to SRS 3.3.0.

One key aspect of the openETCS project is the strict application of formal methods during the development process [70]. This ensures, in contrast to a verbal language specification, a resulting software reference implementation that is error-free and unambiguous. Another key aspect of the openETCS project is the Open Proofs approach as an extension to the Free/Libre Open Source Software (FLOSS) concept [37, p.3]. In contrast to the European Union Public License (EUPL), not only the software is made available to the public, but also all safety cases, documents and tools that are required for the whole development process [66]. The Open Proofs approach supports a pre-competitive cooperation. This concept, also known as Open Innovation, involves external persons with ideas and solutions into the development. This creates knowledge networks that accelerate the research and development. [68]

Since all documents and safety cases are freely accessible to the public and can be evaluated by experts, the quality and safety standards increase. The used methodologies within the openETCS project inevitably lead to a better maintainability and to a standardization that is indispensable for the interoperability aims of ETCS. The final result is a vendor-neutral software

[1] `http://openetcs.org`, accessed 27 September 2015
[2] `http://github.com/openetcs`, accessed 27 September 2015

reference implementation that has a small obsolescence risk and that can be maintained for a long period of time by a big community.

From a financial point of view, the aim of the openETCS project is to reduce the life cycle costs of an ETCS on-board unit. Deutsche Bahn assumes that in the case of success, the costs can be lowered to the price level of current national state-of-the-art solutions. [38]

3 Formal methods

This chapter provides at the beginning a general introduction to formal methods and discusses basic principles. After that, the role of formal methods in the development of railway signaling systems is briefly described. Finally, the SCADE tool, which is used for a formal methods based development within the openETCS project, is introduced.

3.1 Introduction to formal methods

A possible definition of formal methods is given by the military standard Def Stan 00-55 [81]. According to it, a formal method defines a software specification and production method that comprises a collection of precise mathematical notations or computer languages with a formal semantics for specifying the behavior and functionality of a system in a formal manner. This specification can address various phases of the software development life cycle, including specification, design and implementation. Furthermore, a formal method provides tools that are able to guide and assist through the software development life cycle in such a way, so that the resulting system is in a consistent state and works properly. This includes mechanisms in which the functionality of the system can be checked at various phases with formal verification proofs and other properties that can be defined. In addition to that, a formal method is equipped with a methodological framework that has the capability to develop software in a formally verifiable manner. As a consequence, a formal method is more than just a unitary entity. A formal method is rather a collective term for various levels and methods of formal analysis.

Use and relevance. From a historical point of view, formal methods can be seen as an answer to the rapidly increasing complexity in software since the 1960s, which manifested itself in inefficient, low quality, difficult maintainable and non-deliverable software, and led to creation of the term *Software Crisis*. The publication of the *Chaos Report* in the year 1994 [78], in which a high cancellation rate for software project was indicated, illustrated these existing problems clearly.

Formal methods are rarely used in the software industry and are, like already shown in Fig. 2, restricted to specific application domains. However, they become more and more accepted. Formal methods are usually applied in the development of safety-critical software. Roughly speaking, safety-critical software can be seen as software in which errors can result in a potential hazard or loss of predictability or control of a system [15]. In practice, the provision and measurement of safety becomes extremely difficult. This applies particularly to complex software. According to an estimation of [65], approximately 3.3 software errors per thousand lines code exist in large software systems. But what it makes all much more dangerous is that even small errors in safety-critical software can lead to a catastrophe. Formal methods address correctness issues and are a way to improve the confidence in safety-critical software. In practice, formal methods are usually

used in varying degrees in industrial projects. The choice of which level should be used for the development depends on the particular situation. Criteria could be e.g. time, money or personnel. Typically, the application of formal methods can be classified at three levels [39, p.6]:

1. Formal specification
2. Formal specification and semi-formal verification
3. Formal specification and formal verification

As already mentioned in 1.3, railway signaling systems can be considered as one of the most important areas in which formal methods are applied. One of the most important reasons for this success is without a doubt the safety-criticality. Many examples from successful industrial applications exist. In the year 1989, the Parisian RER Line A started its operational service with the automatic train protection system SACEM, which controlled the speed of all trains on the line. The software was written in Modula-2 and contained more than 21,000 lines of code. 63% of the code were regarded as safety-critical. For this part, the formal specification language B [1] was used. For proving the safety properties, automatically generated verification conditions for the code were applied manually. The validation effort for the entire systems was estimated to 90 man-years. [43] Another example is the development of the driverless metro line 14 in Paris, which started its full operation in the year 1998. In this case, B was used for the design of safety-critical parts of the system, concerning an automatic train protection and a control mechanism for platform screen doors. [5] According to [56], no software bugs have been found since the line is in operation.

Formal specification. Put simply, a specification is a detailed description of a product. In the context of software engineering, it can be distinguished between informal and formal specifications. Informal specifications, on the one hand, are basically written in natural language, or represented by diagrams or pseudo code. Their use is a common approach in the software development and they are able to provide in early phases an easy understandable conceptual overview of a system. Formal specifications, on the other hand, use a formal notation or a formal specification language that has a precise mathematical notation for specifying the properties of a system that should be satisfied. Every sentence in the notation or language has a unique mathematical meaning. Thereby, the underlying mathematical concepts are often simple and base e.g. on mathematical logic and set theory.

There is a variety of formal specification languages that are suitable for specifying different kinds of systems. Each of them provides its own style and some of them are designed to be expressive and understandable by humans rather than executable by computers. Formal specification languages have its own underlying mathematical logical frameworks and consist of a notation of models, a notation of sentences that can be used to express properties about such models, and a notation that expresses what it means for a model to satisfy a sentence. Thereby, the models are mathematical abstractions of programs or systems. [39, p.11]

It can be distinguished between five different types of specification languages [4, p.579]. Model-based specification languages provide an abstract definition of a system state and operations that transform the state, but without any explicit representation of concurrency. Examples for this are the specification languages B, Z [51] and the Vienna Definition Language (VDL) as a part of the Vienna Development Method (VDM) [52]. Algebraic-based specification languages give an implicit definition of operations by relating the behavior of different operations without defining a state. Again, concurrency is not supported. Examples are OBJ [34] and PLUSS [6]. Process algebra based solutions use an explicit model of concurrent processes and represent behavior by means of constraints on allowable observable communication between the processes. CSP [46] and CCS [63] are such kind of languages. Specification languages with a logic-based approach use logic to describe the properties of a system. This includes low-level specification of program and system timing behavior. Temporal and interval logics are an example for this [31,53]. Finally, there also exist net-based specification languages. They provide an implicit concurrent model of a system regarding data flow through a network and representing conditions under which data flow can flow from one node in the net to another. Petri Nets [69] and Predicate Transition Nets [86] fall into this category. However, the distinctions between these five types are not always clear. This follows also from the fact that there exist several hybrid variants. The RAISE Specification Language [32] is such an example.

Formal specifications have a number of advantages. Informal specifications are often ambiguous, inconsistent or incomplete. This can lead to errors that are difficult to detect and correct in a later phase of the software development life cycle when the complexity of the system becomes too high. In contrast to that, a formal specification is precise. Therefore, errors in the specification can be found easier and earlier. In addition to that, the overall development costs can stay within limits. However, it must be pointed out that informal specification techniques should not be undersold. Due to their comprehensibility, it is a legitimate approach to combine informal and formal specification to get the best of each. This can be e.g. done in the requirements specification phase in which requirements, which are usually first formulated in an informal manner, are transformed to a formal representation. Moreover, a formal specification is able to improve the understanding of a system, especially in an early design phase. Thoughts can be organized better and as a consequence, a clearer and simpler design is possible. Furthermore, design choices can be better explored. Another advantage of formal specifications is their abstraction, i.e., that implementation details can be omitted that are not relevant for their purpose. How much is omitted depends on the particular context regarding the software development life cycle. Generally, more details are omitted in the earlier phases. Due to abstraction, the focus can lie on what the software should do, and not how it should achieve that. So, it is possible to concentrate on the fundamental details and postpone complex implementation details to later phases of the software development life cycle. This allows a stepwise refinement in which an iterative approach is used (see Fig. 5).

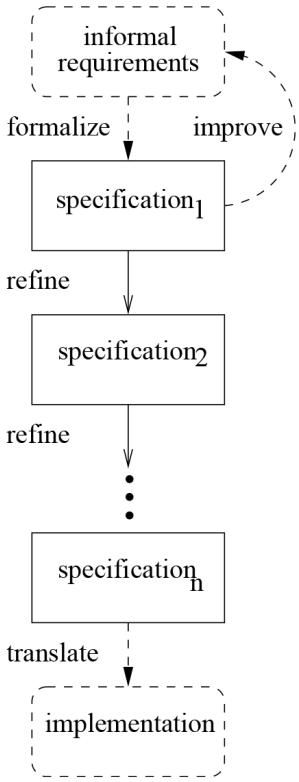


Fig. 5. Stepwise refinement of a formal specification [39, p.9]

Initially starting from informal requirements, the specification becomes more and more detailed in every single iteration step and is verified against the created specification from the previous iteration. The final iteration can than contain enough information to translate the specification into a programming language. This can be e.g. done via automatic code generation tools. Last but not least, abstraction allows several possible implementations that can be formulated with only one generic formal specification, i.e., that several products from different vendors and with different algorithms can exist that fulfill the specification. [39, pp.7-10]

Formal verification. Code inspection and testing are examples for useful techniques that are able to find bugs in software systems. Nevertheless, they may not find all of them. This applies in particular to large software systems in which the number of possible system executions is very large. As a result, it is only feasible to test a small amount of them in practice. Formal verification techniques can solve these problems. Formal verification is a form of analytical reasoning and is used to prove or disprove that the behavior of a software satisfies a given formal specification. In contrast to other techniques, formal verification considers

all possible system situations. Software bugs can be found before the system is implemented. However, it must be considered that formal verification is normally only performed on the formal specification of the system and not on the system itself. As a consequence, formal verification should be seen as a supplement to testing and not as a substitute. [39, p.13f] Theorem proving and model checking are two examples of major state-of-the-art formal verification techniques and are described and compared below.

Theorem proving: In some mathematical logic, a proof of a mathematical statement consists of a sequence of argumentation steps. An argumentation step consists of some premises and a conclusion that can be drawn from the premises, i.e. from statements that are known to be true. Thereby, proof rules for drawing conclusions from premises can be provided by the mathematical logic. The conclusion of an argumentation step can be used as a premise in the following steps. The idea of theorem proving is to construct a mathematical proof for a mathematical statement that has to be true. The statement is known to be true and is said to be a theorem if such a construction results in a successful proof. If a proof is not found, it cannot be concluded that the statement is false. So, on the one hand, it is possible that the statement is false. But, on the other hand, it is also possible that the statement is actually true because the responsible person or proof system was not wise or powerful enough to find a proof.

It can be distinguished between two types of mathematical proofs. Semi-formal proofs are a combination of mathematical formulas and natural language and are subject to interpretation of the reader. Assuming that there exist no inconsistencies, it is in general possible to transform a semi-formal to a formal proof. Formal proofs use in contrast to semi-formal proofs a symbolic language with a precise syntax instead of natural language expressions. Semi-formal proofs can be created much easier and faster compared to formal proofs. Nevertheless, the natural language that is used increases the risk of inconsistencies. So, it can be said that formal proofs are more reliable.

Constructing formal proofs can be time-consuming. Therefore, computer programs exist that can simplify the work. The simplest form are proof checkers. They are programs that automatically can check whether a postulated proof is correct according to the given theorem. MetaMath [62] and Mizar [64] are two examples of proof checkers. Interactive theorem providers are the most common form and can be used to interactively construct a correct proof. KIV [83], PVS [72], Isabelle/HOL [80], ACL2 [85] and Coq [12] can be taken as an example for that. Last but not least, there exist automated theorem prover that automatically search for a proof of a given theorem. The programs Prover9 [84] and SPASS [61] fall into this category. Examples for the application of theorem proving in railway signaling can be found e.g. in [41], [59] and [33]. [39, p.14f]

Model checking: Model checking is a widely used automated technique for complex systems that explores all possible system states in a brute-force manner and verifies some desirable properties. Given a finite-state model of a system and a formal property, it is systematically checked whether this property holds for a

given state of that model. Thereby, the models describe how the state of the system may evolve over time. Fig. 6 shows how the process of model checking is applied.

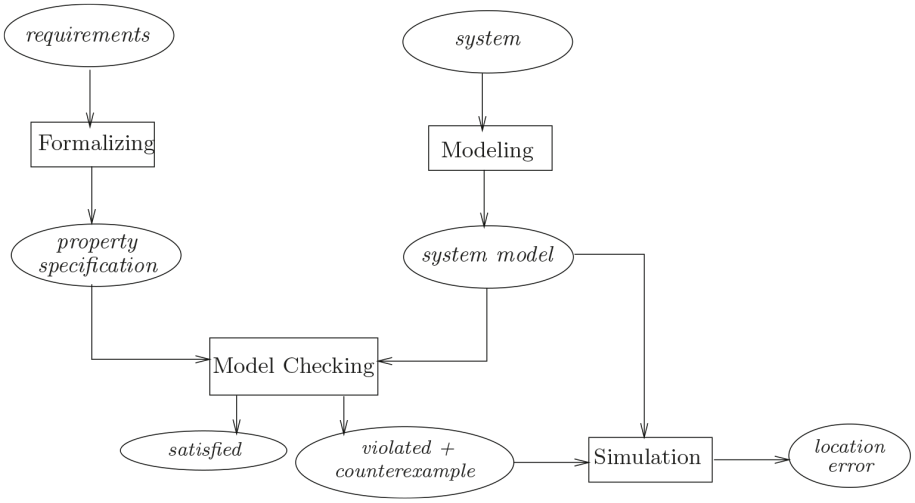


Fig. 6. The process of model checking [3, p.8]

The model checking process consists of the *modeling*, *running* and *analysis* phase. In the modeling phase, a model of the system is created and the informal requirements are formalized that obtain a property specification of these. The models describe the behavior of the system in an unambiguous way and use in most cases finite-state automata, which can be expressed with a model description language. The finite-state automata consist of a finite set of states and transitions. States can contain e.g. information about the current values of variables and the previously executed statement. Transitions describe how the system evolves from one state into another. For improving the quality of the model, simulation can be used prior to model checking. In this way, simple modeling errors can be eliminated and the costly and time-consuming verification effort reduced. The specification of the properties is typically expressed with temporal logics. They are basically an extension of propositional logics and contain operators that can be used to express constraints on how the state of a system may evolve over time. With this, a broad range of relevant system properties can be specified. This includes e.g. functional correctness, reachability, safety, liveness, fairness and real-time properties.

In the running phase, a model checker is used. Assuming that system models have a finite number of states, it explores exhaustively all reachable system states according to the model and checks whether the model truly satisfies the property specification.

Considering the running phase, the model checker can produce three possible outcomes for the analyzing phase. In case that the specified property is valid in the given model, the following property can be checked. If all properties have been checked successfully, the model satisfies the property specification. Another possibility is that the model is not valid, i.e., that a property is not satisfied. In that case, a counterexample is provided that contains a description of a run of the model that leads to a state for which the property specification is not met. This is e.g. the case if the model does not reflect the design of the system. As a consequence, the model must be corrected and the verification repeated with the improved model. Design or property errors occur if there are no inconsistencies between the design and its model. In case of a design error, the verification has failed and the design together with its model has to be improved. In case of a property error, the property that does not reflect the informal requirement that had to be validated must be modified and the verification must be repeated. Last but not least, the physical limit of computer memory can lead to an abort of the verification process. This is especially the case for large software systems in which the model can become too large to be handled by computers with conventional memory size. A remedy is here a reduction and optimization of the model. Examples for the application of model checking in railway signaling can be found e.g. in [40] and [42]. [3, pp.7-16]

Comparing theorem proving and model checking: In contrast to theorem proving, model checking is fully automated and as a result much easier and faster to use. Nevertheless, model checking may not be appropriate for large software systems due to the state-space explosion problem, i.e., that the number of states in the model may exceed the available computer memory. Despite using reduction and optimization techniques for the model as a way out, realistic systems may stay still too large to fit in memory. Currently, state-of the-art model checkers are able to handle state spaces of about 10^8 to 10^9 states. Another disadvantage of model checking is that checking generalizations, such as generic and parametrized systems, is not possible. As a consequence, the correctness of each concrete system must be checked. Theorem proving, on the other hand, can handle generalizations and it can be proved once-and-for-all that any instance of the system is correct. [39, p.19f]

In conclusion, it must be pointed out that model checking can provide a significant increase in the level of confidence of a system design and that it is an effective and fast technique to expose potential design errors. Theorem proving should be preferred in cases where model checking is not applicable due to state space explosion or because the system under consideration cannot be modeled as a finite state system model. However, researchers try to investigate how the best of theorem proving and model checking can be combined and put into practice. The publication of [74] can be taken as an example for that. [3, p.14f]

3.2 Formal methods in railway related certification standards

Several standards for the industrial development of safety-critical software exist that recommend or even prescribe the application of formal methods within the software development life cycle. Examples of such standards are Common Criteria ISO/IEC 15408 [48] for information technology security evaluation, DO-178C [73] for avionics, Def Stan 00-55 and Def Stan 00-56 [82]. For railway signaling systems, EN 50128 [24] is of particular interest.

EN 50128 is a European standard. It concentrates on methods that need to be used in order to provide software that meets the demand for safety integrity at five different levels. It provides a set of requirements with which the development, deployment and maintenance of safety-critical railway systems shall comply. Thereby, techniques are stated that are highly recommended, recommended, mandatory or not recommended. Furthermore, EN 50128 defines requirements concerning the organizational structure, the relationship between organizations and the division of responsibility involved in the development, deployment and maintenance activities.

Safety integrity level. The key concept of EN 50128 is the Safety Integrity Level (SIL). The SIL is a property of a system that is related to the damage that can be caused by a system failure. It is usually apportioned to subsystems and functions at system level in the preliminary risk assessment process and can be also associated to software functions. The SIL consists of five different levels and is defined as a number in the range between 0 and 4. Thereby, SIL 0 gives no safety concern, whereas SIL 4 indicates a high criticality.

Using formal methods. It must be pointed out that the EN 50128 guidelines only classify a wide range of commonly adopted techniques and rate them with respect to the established SIL of the component. So, no precise software development methodologies nor any particular programming techniques are prescribed. Also, no process is prescribed in which formal methods take a role. However, formal methods are rated as recommended (R) or highly recommended (HR) in some phases of the software development life cycle (see Tab. 1):

Table 1. Overview of phases in which the application of formal methods is recommended.

Activity	SIL 0	SIL 1	SIL 2	SIL 3	SIL 4
Software requirements specification	-	R	R	HR	HR
Software architecture	-	R	R	HR	HR
Software design and implementation	-	R	R	HR	HR

3.3 Formal methods based development with SCADE

Formal methods play an important role within the openETCS project. After a careful evaluation of industrial state of the art approaches, SCADE was chosen by the openETCS project leadership as an appropriate tool for a formal methods based development.

SCADE is a software development environment that bases on the formal specification language Lustre. The SCADE environment provides mechanisms to help and assist the formal methods based development of embedded systems and is certified for EN 50128 compliant safety-critical railway signaling systems with SIL 3/4. It is a commonly used tool in the industry and has been e.g. used successfully in important European avionic projects by the former EADS group [54, p.35]. SCADE is composed of several tools such as a graphical editor for designing specifications, a simulator, a model checker and a code generator that transforms graphical specifications to C and ADA code.

The aim here is to give an informal introduction to the formal specification language Lustre and SCADE.

Lustre. Lustre is a strongly-typed, declarative, and data-flow based formal specification language that is designed to write synchronous programs. Based on the synchrony hypothesis, the time is divided into cycles and no reaction time is taken into consideration. Any expression E represents a flow, i.e. an infinite sequence $(e_0, ..., e_n, ...)$ of values [10, p.179]. e_n is the value of E at the n-th execution cycle. At every cycle $n \in \mathbb{N}_{\geq 0}$, the synchronous program takes an input variable i_n from its environment and starts to compute an output variable o_n. During that time, the program is blind to changes in the environment. Considering the fact that the execution time of a cycle is always smaller than the minimum time between two successive inputs, the computation of the output is made instantaneously at the same cycle n. When the output is ready, it is fed back to the environment and the program waits for the start of the next cycle. [54, p.36] Fig. 7 illustrates the behavior of a synchronous program.

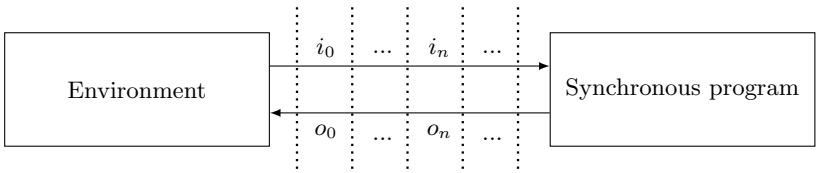


Fig. 7. Behavior of a synchronous program. i_n represents an input value from the environment for the n-th cycle. Its corresponding output value o_n is computed instantaneously at the same cycle.

Language. A Lustre program is structured into nodes. In simple terms, a node can be seen as a Lustre subprogram. Every node defines its output flows as functions of input flows. It receives input variables and computes the corresponding output variables, and possibly local variables, by means of a system of equations. Equations are built using expressions, whereby each expression denotes a flow. An expression E may be a constant, a variable, a condition or the application of an operator $O(i_1, ..., i_k)$ to a k-tuple of inputs. A node `N` is instantiated in Lustre with the syntax **node** $\mathrm{N}(i_1 : \tau_1; ...; i_k : \tau_k)$ **returns** $(o_1 : \theta_1; ...; o_l : \theta_l)$;. If $E_1, ..., E_k$ are expressions of type $\tau_1, ..., \tau_k$, then the instantiation $\mathtt{N}(E_1, ..., E_k)$ is an expression of type $(\theta_1, ..., \theta_l)$ whose value is the tuple $(o_{1_n}, ..., o_{l_n})$ that is computed by the node from the input parameters $E_1, ..., E_k$ in the current execution cycle n. [10, p.180f]

A Lustre program can be represented as a directed graph, also called operator network (see Fig. 8). An operator network shows the data-flow through a set of operators in a graphical manner. An operator either corresponds to a node or to a basic computation.

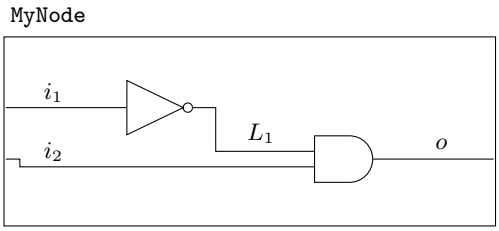


Fig. 8. Example of an operator network

From a mathematical viewpoint, an operator network $\aleph$ is a multi-entry directed graph that consists of a set of N operators and a set $\mathrm{E} \subseteq N \times N$ of directed edges that connect operators. An operator network contains as many entry edges as input variables exist. This applies respectively to the output edges and output variables. Entry edges have no inward and exit edges have no outward operators. Furthermore, there also exist internal edges that have one inward and at least one outward operator. An edge e_2 is called successor of the edge e_1 if an operator inside the network contains e_1 as an input and e_2 as an output. Every operator network defines paths. Thereby, a path is a finite sequence $\langle e_0, ..., e_n \rangle$ with $\forall i \in [0, n-1] : \langle e_i, e_{i+1} \rangle \in \aleph$, i.e., that e_{i+1} is a successor of e_i. [54, p.36f] Looking again to Fig. 8, the operator network contains two entry edges, one exit edge, two logical operators (**not**, **and**) and the elementary paths $(\langle i_1, L_1, o \rangle, \langle i_2, o \rangle)$.

Operators. Lustre is limited in its language features. Iterations or recursive calls are e.g. not supported. As a consequence, it could be argued that the language is not practically usable for solving challenging software engineering problems.

However, powerful temporal operators are provided. Furthermore, it is possible to call complex external functions that are written in a host language such as C or ADA. Below, an overview of some of the most characteristic Lustre operators is given [10, p.179f]:

- **pre**(X): the temporal operator **pre**(X) returns the value of its previous cycle $n-1$:

$$\textbf{pre}(\mathrm{X}) = (\underbrace{\texttt{nil}}_{\text{undefined}}, x_0, ..., x_{n-1}, ...)$$

- X−>Y: the temporal *followed by* operator X−>Y is similar to **pre**(X) but allows to set an initialization value X $\neq$ `nil` for the first execution cycle n:

$$\mathrm{X}{->}\mathrm{Y} = (x_0, y_0, ..., y_{n-1}, ...)$$

- E **when** B: Lustre computes every cycle n the n-th value of every variable. However, there may be some reasons where it becomes necessary to compute a value only under certain conditions and not at every cycle. Lustre provides the sampling operator **when** for solving this concern. If E is an expression and B a boolean expression, then E **when** B takes only the sequence of those values from E when B is `true`. So, it is e.g. possible to interact with different parts of a program that have among each other a different cycle time:

$$\begin{aligned}
\mathrm{E} &= (\ e_0 \quad e_1 \quad e_2 \quad e_3 \quad e_4 \quad e_5 \quad ...) \\
\mathrm{B} &= (\ \texttt{true} \ \texttt{false} \ \texttt{true} \ \texttt{true} \ \texttt{false} \ \texttt{false} \ ...) \\
\mathrm{X} = \mathrm{E}\ \textbf{when}\ \mathrm{B} &= (\ x_0 = e_0 \qquad x_1 = e_2 \ x_2 = e_3 \qquad ...)
\end{aligned}$$

- **current**(E **when** B): the operator **current**(E **when** B) allows operations over variables of different cycle times. If B is `false`, then the value taken by E at the last cycle when B was `true` is returned:

$$\begin{aligned}
\mathrm{E} &= (\ e_0 \quad e_1 \quad e_2 \quad e_3 \quad e_4 \quad e_5 \quad ...) \\
\mathrm{B} &= (\ \texttt{true} \ \texttt{false} \ \texttt{true} \ \texttt{true} \ \texttt{false} \ \texttt{false} \ ...) \\
\mathrm{X} = \mathrm{E}\ \textbf{when}\ \mathrm{B} &= (\ x_0 = e_0 \qquad x_1 = e_2 \ x_2 = e_3 \qquad ...) \\
\mathrm{Y} = \textbf{current}(\mathrm{X}) &= (\ y_0 = x_0 \ y_1 = x_0 \ y_2 = x_1 \ y_3 = x_2 \ y_4 = x_2 \ y_5 = x_2 \ ...)
\end{aligned}$$

Formal verification. Formal verification is done in Lustre by using synchronous observers. Synchronous observers are operators that are used to express safety properties that should always hold. The idea behind synchronous observers is to construct an observer S' by composition of the observed system S and a safety property P (see Fig. 9). S is considered as a black box and only its inputs and outputs are used for the formulation of P. S' contains as an output only a single boolean value of P. This output states if the property holds at each cycle. The task of the formal verification consists in proving that the output of S' is always true. [55, p.471]

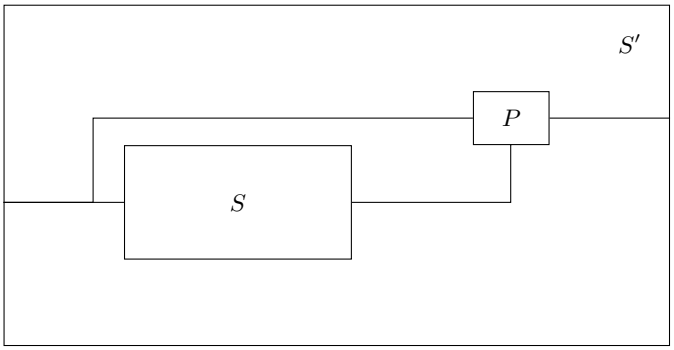


Fig. 9. Synchronous observer

SCADE. The SCADE language is a graphical, data flow based formal specification language that can be translated to Lustre. It is built on formal foundations and has a strong deterministic behavior. SCADE enables a model-driven software development approach and simplifies the practical application of formal methods in the whole development process. SCADE extends the language features of Lustre in many ways. New operators, types and concepts were introduced. Furthermore, the language was in some parts designed to be more restrictive than Lustre. In the following, an overview of some characteristic SCADE features is given [16].

Hierarchical state-machines. SCADE provides in addition to the data-flow formalism of Lustre, control flow mechanisms that are realized with hierarchical state-machines (see Fig. 10).

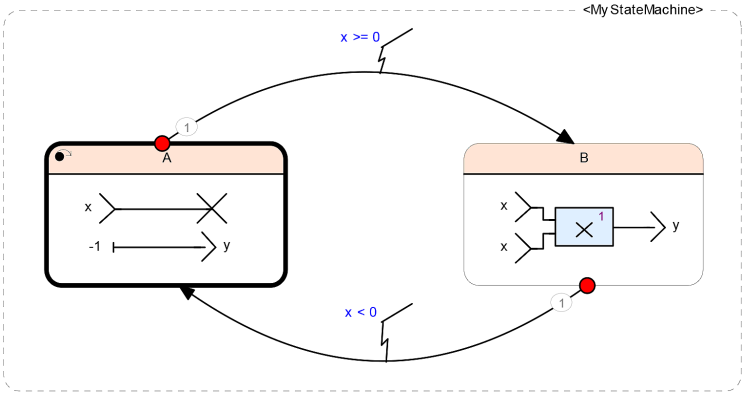


Fig. 10. Example of a hierarchical state-machine in SCADE. A represents the start state and computes the output $y = -1$, state B computes $y = x^2$. Two strong state transitions exist. Both of them have the highest priority 1 and the transition conditions $x \geq 0$ and $x < 0$.

Data-flow and control flow can be mixed together without violating the basic data-flow principle. Every state can contain nested sub-states and only one active super-state per cycle. Like in a finite state machine, one start state and zero or more final states exist. A transition between two states is only taken when the transition condition is met. When a transition is fired, actions such as signals can be emitted. Furthermore, a state can have more than one outgoing state transition. However, it is only allowed to take one outgoing state transition at the same time. As a consequence, every outgoing state transition has a priority. SCADE offers three different kinds of state transitions. When a strong transition is fired, the target state becomes active within the cycle. When a weak transition is fired, the target state becomes active in the next cycle but the transition actions are activated within the current cycle. Synchronization transitions are used for synchronizing parallel state-machines.

Iterators. SCADE provides with several predefined iterators scheme safe loop constructs that make the processing of arrays more effective. Iterators are higher order operators and can be applied to every node. An example for an iterator is the **map** operator. Given a node `Sum` that takes as an input two integer variables a, b and computes the output $c = a + b$, a pointwise sum of two integer arrays can be expressed as shown in Fig. 11.

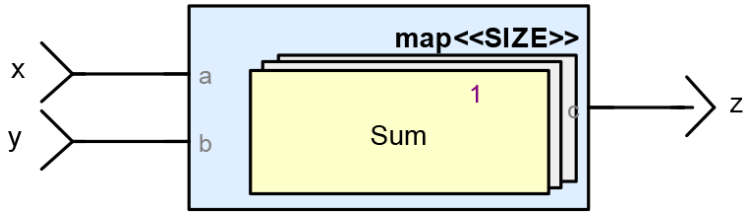


Fig. 11. Example of a **map** iterator in SCADE. x, y, z are three integer arrays with the same length `SIZE` > 0. The output is $z = \{x[0] + y[0], ..., x[\texttt{SIZE} - 1] + y[\texttt{SIZE-1}]\}$.

Activation conditions. Sampling operations can be performed easier in SCADE than with the **when** operator in Lustre. Thereby, a boolean flow can be attached with an initialization value to a node call as an activation condition. The node is only active when the activation condition is `true`, otherwise the output of the node keeps its last known value. Fig. 12 shows the operator of Fig. 10 in combination with an activation condition.

Design by contract. SCADE supports a design by contract approach. Thereby, a contract is a specification of expectations before applying a node and, supposing they are verified, of some properties on the results of this application. This contract is made of a pair of observers. The **assume** operator observes inputs and the past of outputs, the **guarantee** operator observes inputs and outputs.

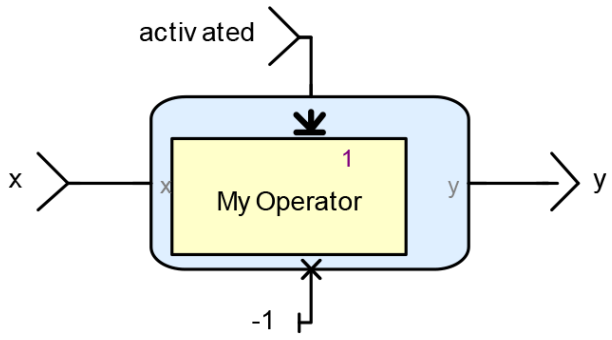


Fig. 12. Example of a SCADE operator with an activation condition. The node `MyOperator` is only active when the boolean variable `activated` is `true`. In case that in the first execution cycle the activation condition is set to `false`, the initialization value -1 is returned.

Sensors. A sensor defines a global flow that allows to connect an application to its environment by allowing any node to access some global input. It remains stable during all the cycle computation and has the ability to be updated between each cycle.

Packages. SCADE provides a package mechanism. This feature, which is provided by many programming languages, helps to make the design of the software more clearly and facilitates the usage of libraries.

Polymorphism. SCADE supports generic programming and provides a way to define polymorphic nodes, i.e., that a node can be defined independently from the type of its input.

4 The RBC

The RBC is a trackside, computer-based subsystem of ETCS. It acts as a complementary protection to enforce the safety of train movements on track sections and is responsible for a bidirectional communication with trains. On the basis of information that is received from external trackside systems, such as interlockings, and on the basis of information that is exchanged with the EVC of a train, radio messages are sent to this train via a GSM-R connection. The main objective of these messages is to provide movement authorities to allow safe train runs on unoccupied track sections under the responsibility of the RBC without causing operational hazards. [79, p.32]. In the following, the functionality of the RBC is described to provide basic knowledge for the formal specification in the next chapter.

4.1 Operating levels.

There are five different functional levels in ETCS. Each of them determines a usable range of functions. The level depends on the trackside equipment that activates the level on the train with the corresponding functionality. The RBC is an integral part in ETCS Level 2 and 3 (see Fig. 13). ETCS Level 3 works similarly to ETCS Level 2 but is out of the scope of the openETCS project and therefore not taken into further consideration.

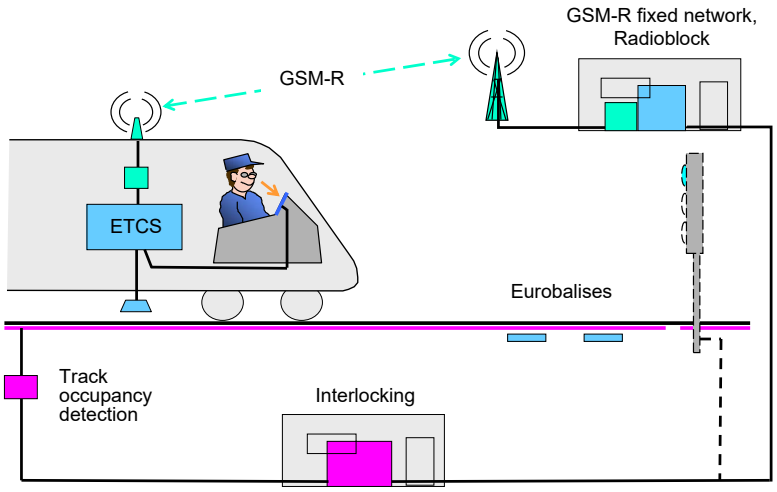


Fig. 13. ETCS Level 2 [87, p.15]

The RBC is designed as an overlay system to an existing track infrastructure. Track clearance detection and train integrity is ensured by non-ETCS trackside components such as interlockings and track occupation devices (track circuits,

axle counters etc.). The ETCS-based trackside equipment consists of the RBC and fixed Eurobalises that are installed on the track. [79, p.49]

An Eurobalise is an inductively coupled, unidirectional transmission device (see Fig. 14). If it is passed by a train, then it generates a magnetic field that is picked up by an antenna that is located under the train. Thus, information is transmitted to the train. [79, p.176f]

Fig. 14. Eurobalise [76]

An Eurobalise forms always a balise group that consist of between one and eight Eurobalises [27, 3.4.1]. A balise group serves mainly as a milestone in ETCS Level 2 [79, p.49]. The train, which has to periodically report its position to the RBC, refers in its position report always to its last relevant balise group (LRBG) and the distance to it.

The RBC uses GSM-R for a bidirectional communication with a train. The continuous monitoring of the train permits train runs with more than 160 $\frac{\text{km}}{\text{h}}$ and allows rail operations without lineside signals. The RBC contains a routemap in which all important track-related information is stored. This includes e.g. information about the track condition and the locations of the balise groups. Building on this information and the information that is received from the interlocking, the RBC sends movement authorities to a train. [79, p.49] This information is shown to the driver on a special device, the so-called Driver Machine Interface (DMI). The train driver is independent from national signaling rules and does not have to take care about lineside signals.

4.2 Radio messages.

It can be distinguished between radio-based track to train and radio-based train to track messages. Thereby, every message includes user data on application level and protocol data that depends on the transmission medium, in this case GSM-R. For transmitting information on application level, the ETCS language is used. It defines variables, packets and the structure of radio messages. All definitions take a bitwise representation as a basis and are independent of the transmission medium. [79, p.91]

Variables. Variables are used to encode single data values that cannot be split into minor units. Every variable has a unique name and meaning. It can contain in addition to standard, also special and spare values. [79, p.90] A definition of all ETCS variables can be found in [27, 7.5]. An example of a variable with the name *Q_DIR* is shown in Tab. 2. This variable specifies for which running direction of the train the transmitted information is valid.

Table 2. Example of a variable definition

Name	Validity direction of transmitted data		
Description	Qualifier to indicate the relevant validity direction of transmitted data, with reference to directionality of the balise group sending the information or to directionality of the LRBG, in case of information sent via radio.		
Length of variable	**Min. Value**	**Max. Value**	**Resolution/formula**
2 bits			
Special/Reserved Values	00	Reverse	
	01	Nominal	
	10	Both directions	
	11	Spare	

Packets. Packets are multiple variables that are grouped into a single unit with a defined internal structure. It can be distinguished between track to train and train to track packets.

A track to train packet consists of a header and a well-defined set of variables. The header contains a unique packet identification number [27, 7.5.1.93], information about the packet length in bits [27, 7.5.1.49], the orientation information *Q_DIR* and optionally a distance scale [27, 7.5.1.129]. Train to track and track to train packets are from a structural point of view equal, with the exception that the running direction *Q_DIR* is unused. The structure of a track to train packet is shown in Tab. 3.

Table 3. Structure of a track to train packet [79, p.91]

Variable	**Description**
NID_PACKET	Packet identification number that allows the receiving equipment to identify the packet data that follows.
Q_DIR	See definition of *Q_DIR* in Tab. 2
L_PACKET	Number of bits in the packet
Q_SCALE (optional)	Qualifier to indicate the scale that is used for describing all distances inside the packet. This variable is only used if distance information is contained within the packet.
Information	Well defined set of variables

In ETCS version 3.3.0, 52 track to train and 8 train to track packets are supported. One packet is used for both directions. It has to be mentioned, however, that not all of them are used by the RBC. An overview of all supported packets is given in [27, 7.4]. Packets that are relevant for the master's thesis are listed in section B (see appendix).

Message. A radio message is composed of a header, predefined variables and packets. Some messages can contain optional packets as defined in [27, 8.4.4.4.1]. The maximum message length on application level is limited to 500 bytes [27, 4.2.2]. To obtain an integer number of bytes, padding is added at the end of every radio message if it is required. A definition of all available radio messages can be found in [27, 8.5]. An overview of radio messages that are relevant for this thesis can be found in the appendix. The structure of a track to train and train to track message is shown in Tab. 4 and 5.

Table 4. Structure of a radio-based track to train message [79, p.93]

Variable	**Description**
NID_MESSAGE	Message identification number
L_MESSAGE	Length of message in bits (without padding)
T_TRAIN	Timestamp of RBC
M_ACK	Indicates whether the track to train message must be acknowledged by the train
NID_LRBG	Identification number of the LRBG
Variables as required by *NID_MESSAGE*	If needed, additional variables can be sent that are not included in a packet.
Packets as required by *NID_MESSAGE*	If needed
Optional packets	If needed (see [27, 8.4.4.4.1])
Padding	If required

Table 5. Structure of a radio-based train to track message [79, p.94]

Variable	**Description**
NID_MESSAGE	Message identification number
L_MESSAGE	Length of message in bits (without padding)
T_TRAIN	Timestamp of the train
NID_ENGINE	Unique identification number of the train
Variables as required by *NID_MESSAGE*	If needed, additional variables can be sent that are not included in a packet.
Packet 0 or 1	Contains a position report of the train
Packets as required by *NID_MESSAGE*	Only for train to track message 129
Optional packets	If needed (see [27, 8.4.4.4.1])
Padding	If required

4.3 Movement authorities.

The trackside equipment, interlocking and RBC are responsible for managing safe train movements depending on the train location and the track design. The permission for a train movement is traditionally shown as a signal aspect in a color light signal that is located besides the railway line (see Fig. 15). [79, p.202]

Fig. 15. German Kombinationsignal (Ks signal) with the signal aspect "Hp 0: Stop" [71]

A movement authority is a translation of a signal aspect. On the one hand, the translation can be done directly from a color light signal. On the other hand, the translation can be done from a radio-based track to train message message that

is generated by a RBC. Such kind of a movement authority contains always the distance to the next track section of interest and the track description up to this location. If there is no permission to enter this section, then there is a so-called End Of Authority (EoA) and the target speed is restricted to 0 $\frac{\text{km}}{\text{h}}$ (see Fig. 16). Otherwise, there is a Limit of Authority (LoA) and the target speed is restricted to a value greater than 0 $\frac{\text{km}}{\text{h}}$. [79, p.202f]

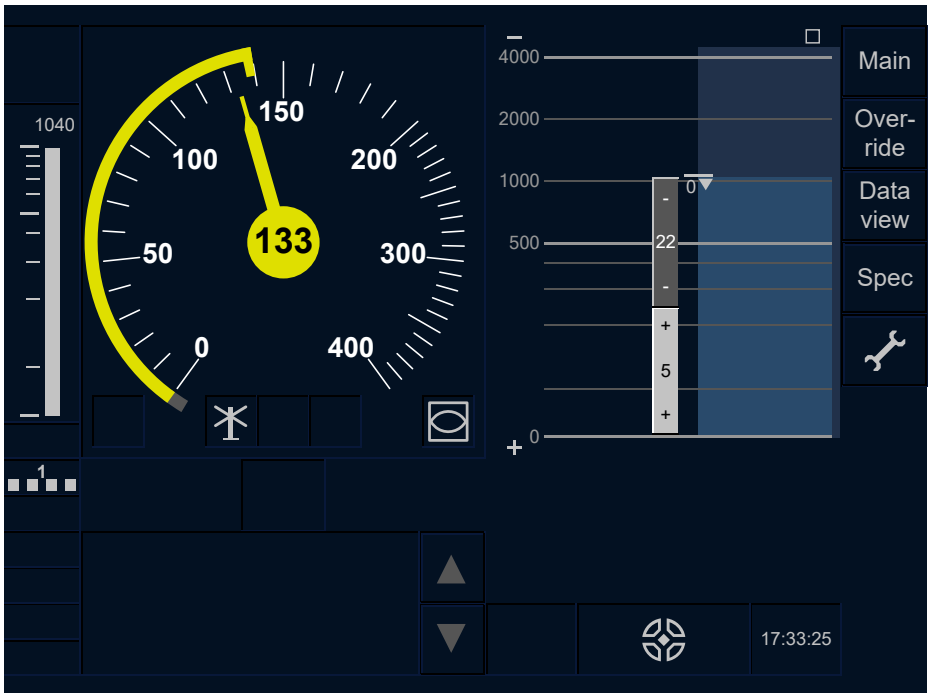


Fig. 16. Visualization of an EoA on a DMI. The current speed of the train is 133 $\frac{\text{km}}{\text{h}}$. An EoA is announced in less than 1040 m. [28, p.94]

5 Formal specification of the RBC in SCADE

In this chapter, it is described how the RBC model is created. First of all, scenarios are analyzed that can appear during the operation of the RBC. Building up on these scenarios, a formal specification of the RBC is created with the SCADE environment.

5.1 Scenarios

One of the main objectives of the openETCS project is to prove the formalization of the EVC with a successful train run on the Dutch railway line Utrecht - Amsterdam. This line is equipped with ETCS Level 2 and contains one RBC that operates with the older ETCS version 2.3.0.

The main focus of interest lies on certain interactions that appear in regular operation between the rolling stock and the RBC. After a careful analysis of the event recorder, also called Juridical Recording Unit (JRU), from a German ICE 3 high speed train that performed several train runs on the target track, several top-priority scenarios were identified that have to be supported by the RBC model. Below, an overview of all relevant scenarios is given.

Establishing a communication session. An established communication session [27, 3.5.3] is a prerequisite for any message-based interaction on application level between a train and a RBC. The process of establishing a communication session is shown in Fig. 17.

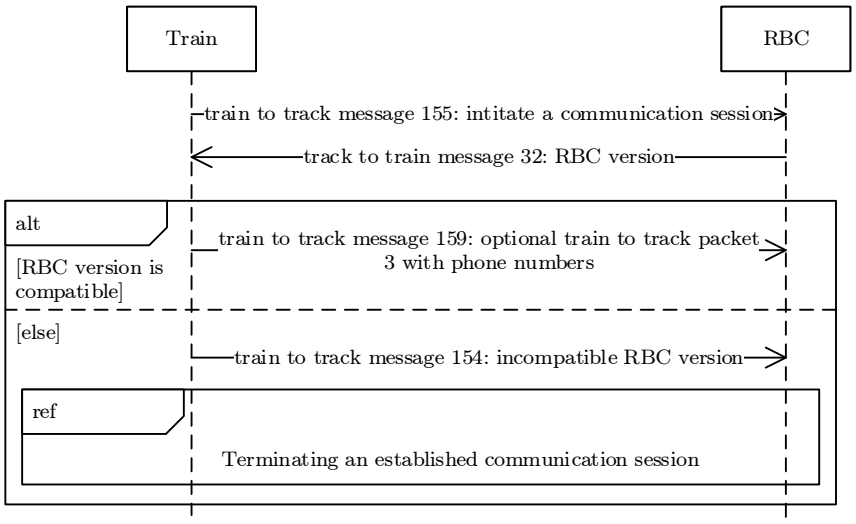


Fig. 17. Sequence diagram of establishing a communication session

Assuming that no communication session is already established or being established at the moment, the RBC receives from a train the train to track message 155 [27, 8.6.13], which indicates that the train wants to initiate a communication session. The RBC answers on this request and sends its operating version to the train with the track to train message 32 [27, 8.7.12].

Afterwards, the RBC receives a session established report that is provided with the train to track message 154 or 159. In both cases, the communication session is considered as established. If the operating version is compatible, then the train to track message 159 [27, 8.6.17] is received. This message can contain the optional train to track packet 3, in which phone numbers are stored that can be used for a direct communication between the train crew and the responsible train dispatcher. If the operating version is incompatible, then the RBC is notified with the train to track message 154 [27, 8.6.12]. In this case, however, the RBC has to wait until it receives from the train a request to terminate the established communication session (see Fig. 18).

Terminating an established communication session. An established communication session can be only terminated by a request from a train. This process is shown in Fig. 18.

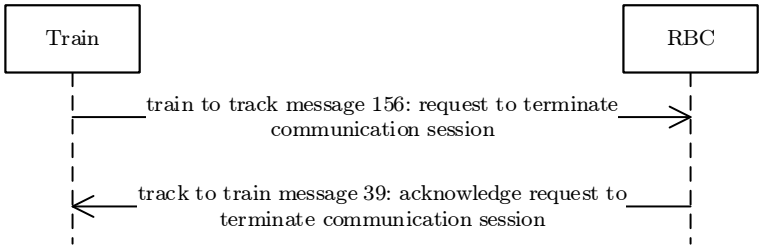


Fig. 18. Sequence diagram of terminating an established communication session

According to [27, 3.5.5], the RBC receives with the train to track message 156 [27, 8.6.14] a request to terminate the established communication session. It acknowledges this request with the track to train message 39 [27, 8.7.17] and the communication session is considered as terminated.

Receiving validated train data. The validated train data contains parameters such as the train length, number of axles, maximum allowed speed and the supported voltage systems. The RBC, which has knowledge about the track layout and all technical restrictions that are associated with it, can decide on the basis of this train data if the train is permitted to run on the track or not. Considering the formal specification, no verification of the validated train data has to be performed, i.e., that the train is always accepted by the RBC. Fig. 19 shows the process of receiving the validated train data.

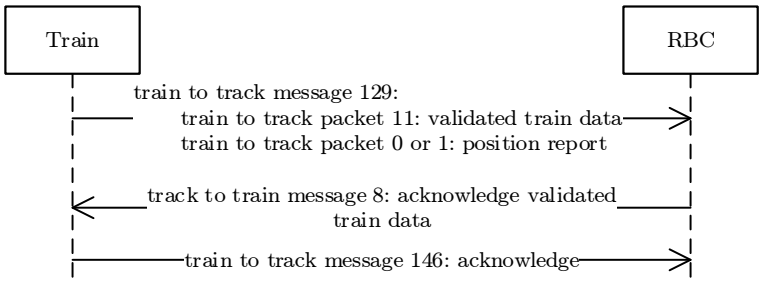


Fig. 19. Sequence diagram of receiving validated train data

Taking a closer look to the interaction between the train and the RBC, the RBC receives as a first step the train to track message 129 [27, 8.6.1], in which the validated train data is contained in the train to track packet 11 [27, 7.4.3.5]. In addition to that, a first train position report is provided by the train to track packet 0 or 1 [27, 7.4.3.1-7.4.3.2], which contains parameters from the odometry such as the LRBG, the distance to it and the current velocity of the train.

The RBC sends with the track to train message 8 [27, 8.7.4] an acknowledgment to the train, which states that the RBC successfully verified the validated train data and that the train has the permission to run on the track. After that, it receives with the train to track message 146 [27, 8.6.7] an acknowledgment from the train and the process of receiving the validated train data can be considered as finished.

Receiving train position report. The train reports its position to the RBC with the train to track packet 0 or 1. This is done by periodically sending the train to track message 136 [27, 8.6.4] as shown in Fig. 20.

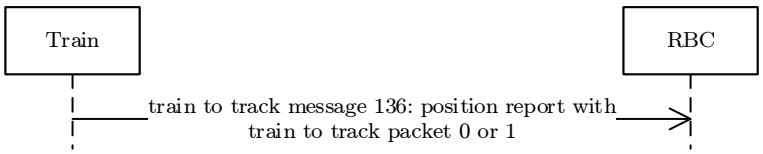


Fig. 20. Sequence diagram of receiving a train position report

Sending a general message. The RBC can send in certain situations the track to train message 24 [27, 8.7.9], also called general message, to a train (see Fig. 21).

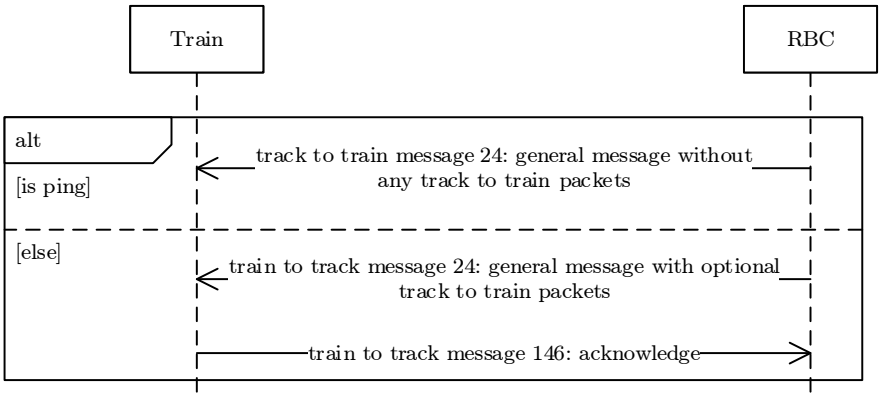


Fig. 21. Sequence diagram of sending a general message

According to [27, 3.16.3.4], a train has to periodically receive from the RBC a consistent track to train message within a fixed contact time that depends on the national operating rules and that may differ from one country to another. In this case, the general message acts as a ping that is sent periodically to the train to satisfy this required timing constraint. This ping message has not to be acknowledged and contains no track to train packets.

A general message can also serve as a container for track to train packets. In this case, it contains at least one track to train packet as defined in [27, 8.4.4.4.1]. This kind of general message has to be acknowledged by the train.

Granting a movement authority. Depending on decisions from the interlocking, a movement authority can be granted by the RBC to a train without waiting for a request from it (see Fig. 22).

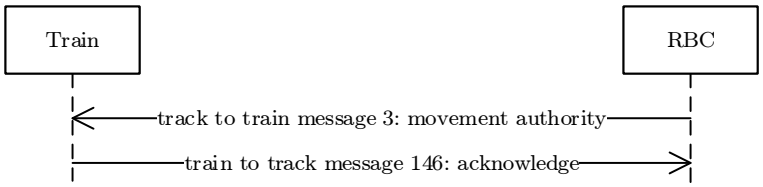


Fig. 22. Sequence diagram of granting a movement authority

As a first step, the RBC grants a movement authority to the train with the delivery of the track to train message 3 [27, 8.7.2], which contains information such as the section length, the permitted target speed at the end of the section, the overlap and existing danger points. The train receives the movement authority and acknowledges it with the train to track message 146.

Granting a movement authority request. A movement authority can also be granted to a train after a request with the train to track message 132 [27, 8.6.3] has been received from it. This process is shown in Fig. 23.

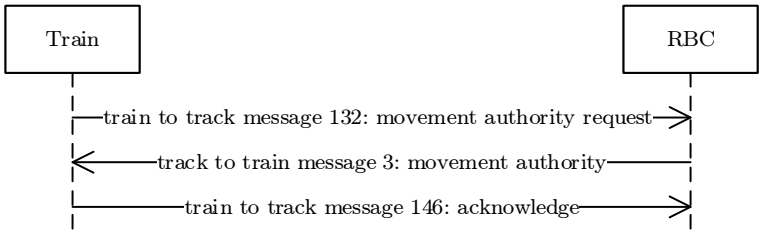


Fig. 23. Sequence diagram of granting a movement authority request

Requesting a conditional emergency stop. Let us consider that train T_A received a movement authority. Due to special circumstances, another train T_B should be prioritized and get a route assigned that stays in conflict with the route of train T_A. For resolving this conflict, the movement authority of train T_A has to be shortened to a certain safe location. The RBC can try to force such a constellation with a conditional emergency stop (see Fig. 24).

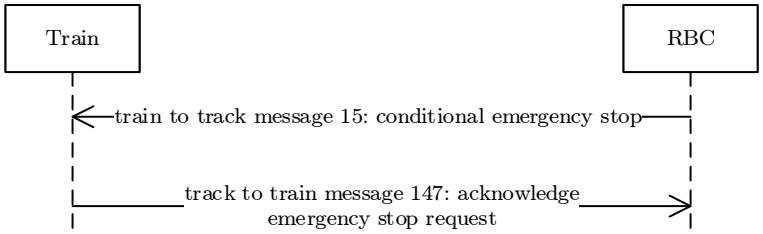


Fig. 24. Sequence diagram of requesting a conditional emergency stop

Considering the interaction steps, the RBC requests from the train with the train to track message 15 [27, 8.7.6] to perform a conditional emergency stop [27, 3.10.2]. This request always contains information that refer to a stop location of the train. If the train has not yet passed this stop location, then the conditional emergency stop request is accepted by the train and its movement authority is shortened to that position. Otherwise, the request is rejected. In both cases, the RBC is notified with the acknowledgment message 147 [27, 8.6.8], which contains the decision of the train and its current position.

5.2 Formal specification.

Having described the operational scenarios, the next task is to formally specify the RBC in SCADE. The basis for all further discussions is the RBC model with the GitHub commit number *bfeac74f1f0422bca713ed0dc028b645dbe629a1*[3]. Because this is a big and complex project, only the most important concepts are described.

Radio messages. Radio messages are an essential part in the interaction between a train and a RBC. In the following, fundamental concepts concerning the formal specification of radio message related features are discussed.

Consistency of train to track messages. Inconsistent train to track messages may pose a safety hazard for the RBC. As a consequence, they must be detected and excluded in any case from further processing.

[27, 3.16.3.1.1] defines consistency criteria for radio messages. First of all, all required variables have to be set as defined in [27, 8.6]. In addition to that, every variable must have a valid range as defined in [27, 7.5.1]. Moreover, the sequence of received radio messages has to be valid. Every radio message contains a timestamp [27, 7.5.1.154] that represents the time at which the radio message was sent. Thereby, a sequence can be considered as valid if the timestamp of the current received radio message is greater than the timestamp of the previous received consistent radio message. To provide a better understanding, a possible sequence of received train to track messages is shown in Fig. 25.

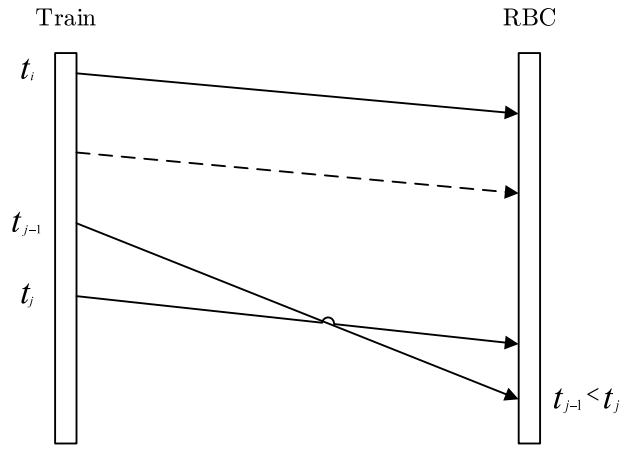


Fig. 25. Example of a sequence of received train to track messages. The last received train to track message causes a sequence error within the RBC because its timestamp t_{j-1} is smaller than the timestamp t_j of the previous received train to track message.

[3] `http://git.io/vnYGm`, accessed 18 September 2015

A consistency check is immediately performed within the RBC model after a train to track message has been received. This is done within the operator *RadioTrainTrackMessageConsistencyChecker__IsConsistentMsg* (see Fig. 26).

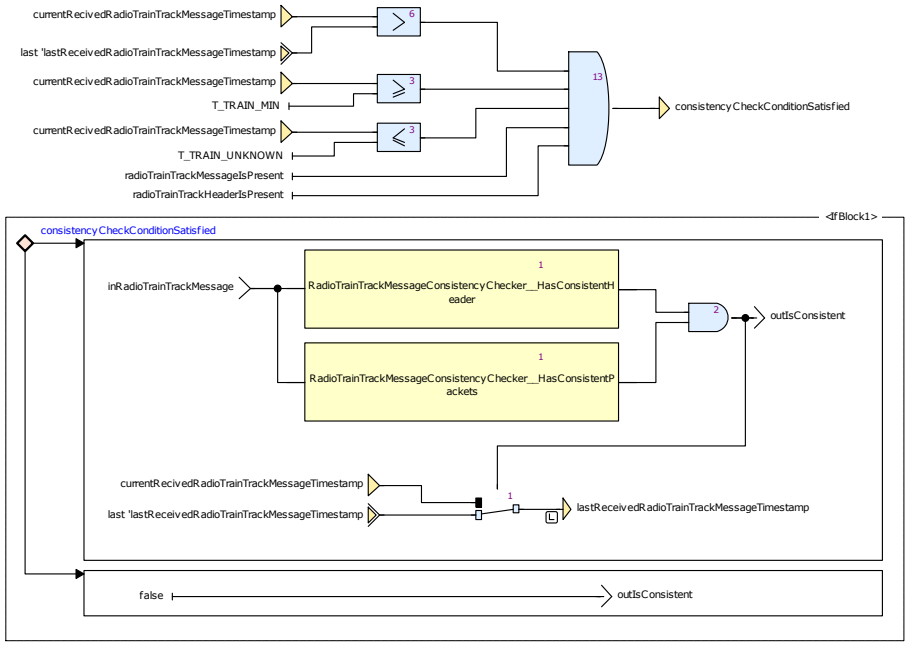


Fig. 26. Consistency check of a train to track message

At the beginning, a sequence check is performed. It is checked if the timestamp of the current received train to track message, which is represented by the local variable *currentRecivedRadioTrainTrackMessageTimestamp*, is greater than the timestamp of the previous received consistent train to track message, which is represented by the local variable *lastReceivedRadioTrainTrackMessageTimestamp*. The result of this check is written into the local variable *consistencyCheckConditionSatisfied*. If it takes the value *true*, then the sequence is valid and the if-branch is entered. Inside it, the consistency of the variables that are contained in the message header and message packets is checked. If this check has been successful, then the received train to track message is consistent and the value of the internal timestamp clock is assigned to the local variable *lastReceivedRadioTrainTrackMessageTimestamp*. Otherwise, the train to track message is inconsistent and the local variable *currentRecivedRadioTrainTrackMessageTimestamp* holds its last known value.

In accordance with [27, 3.16.3.1.1.1], every inconsistent train to track message is rejected immediately. This behavior ensures that all succeeding operator calls always perform operations on a consistent train to track message.

Creating track to train messages. An external SCADE library[4] provides mechanisms to create track to train messages. The library uses, instead of a bitwise representation as defined in [27], an integer-based representation of all variables that are contained in a track to train message, i.e. every variable is stored as an integer. This representation is, despite of small nuances, very close to the original definition because a bit sequence of a single variable can be always represented as an integer. Fig. 27 shows the structure of a track to train message in SCADE.

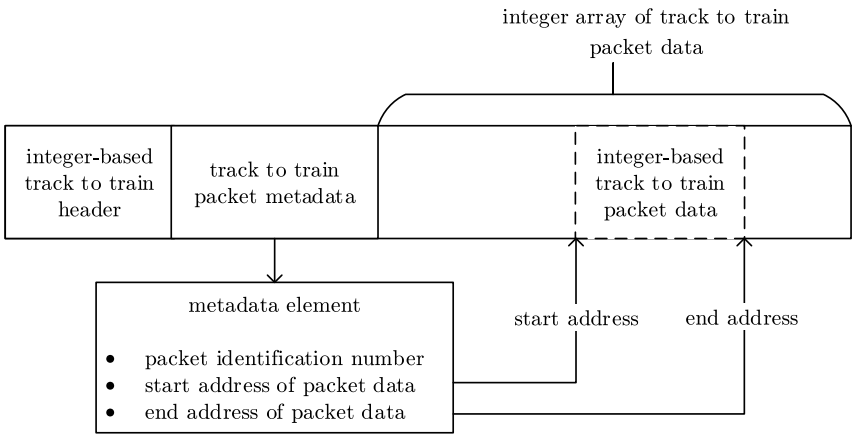


Fig. 27. Structure of a track to train message in SCADE

Every track to train message in SCADE consists of a header, track to train packet related metadata and an integer array that stores the packet data. The header contains all track to train header variables in an integer-based representation. The packet-related metadata consists of a fixed number of metadata elements. Each metadata element contains with the packet identification number information about which kind of track to train packet it describes. A pointer to a start and end address to the integer array with the packet data defines which range is reserved for this packet. This information is needed for the later extraction of single packets from the message.

The representation of track to train messages has disadvantages concerning the usage of ETCS variables that were defined[5,6] by the project team within the SCADE environment. Many of these variables are not integers. An example is the operating version 2.3.0 that is represented in SCADE as an enumeration item that corresponds to the enumeration type *M_VERSION*. The specification, however, does not define enumerations. According to [27, 7.5.1.79], the operating version 2.3.0 is represented by the variable *M_VERSION* that has the

[4] `http://git.io/vZAuS`, accessed 16 September 2015
[5] `http://git.io/vZhn9`, accessed 16 September 2015
[6] `http://git.io/vZhck`, accessed 16 September 2015

bit sequence $(00010000)_2$. Its corresponding integer value that is needed for the creation of a track to train message would be 16. In this case, however, an integer value and an enumeration item are semantically not the same. Therefore, an appropriate solution is needed to resolve incompatibilities between the different data types in the model.

To create track to train messages, non-integer data types need to be converted to an integer representation first. The RBC model provides within the package *RBC_Messaging_Pkg::RBC_RadioTrackTrainFactory_Pkg::* operators that are able to create in an easy way track to train messages while hiding the complexity of the conversion operations. Fig. 28 shows, as an example, how a general message with the optional track to train packets 57 and 58 is created by applying these operators.

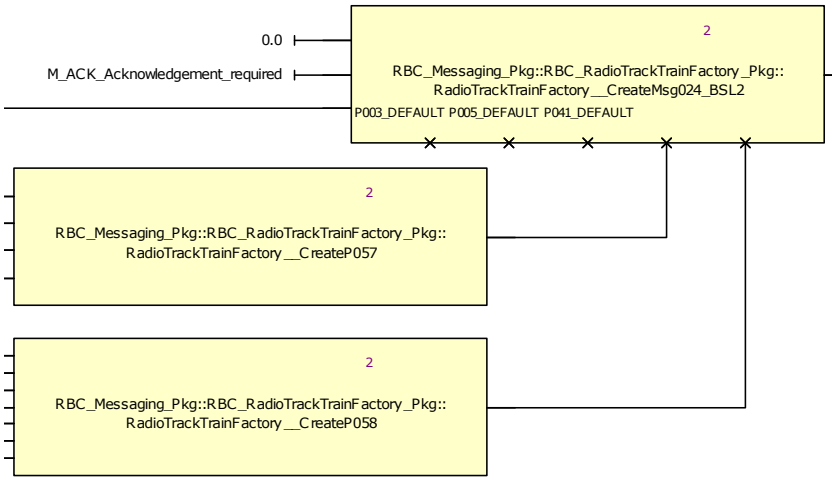


Fig. 28. Creating a general message

The operators *RadioTrackTrainFactory__CreateP057* and *RadioTrackTrainFactory__CreateP058* create, independently of their input data types, an integer-based representation of the track to train packets 57 and 58. These packets are passed to the operator *RadioTrackTrainFactory__CreateMsg024_BSL2* as hidden inputs. A hidden input represents thereby an usual input that is arranged at the bottom of an operator box and that has to be instantiated with a default value if no data flow goes in. Because the optional track to train packets 3, 5 and 41 shall not be part of the general message, the first three hidden inputs are unused and therefore instantiated with default values. The non-hidden inputs are used within the operator for the creation of an integer-based representation of the track to train header. The integer-based track to train header and the integer-based track to train packets are merged together. The result is finally a track to train message that contains the necessary metadata.

Buffering track to train messages. It is possible that more than one track to train message is created within one cycle. However, only one track to train message can be sent per cycle to a train due to the interface definitions of the EVC model. Therefore, a mechanism is needed that is able to store and manage multiple track to train messages.

The RBC model uses a first in first out (FIFO) buffer to accomplish this. If a track to train message is created, then it is stored inside this buffer. At the end of every cycle, one track to train message is taken away from it, timestamped and sent. The remaining track to train messages are sent in the next cycles.

Timestamping of track to train messages. Every sent radio message must contain a valid timestamp. Unfortunately, the handling of time in SCADE is not as easy as it might appear at first sight. The reason for this is that SCADE uses a cycle-based approach and that the cycle time is heavily dependent from the target hardware.

The RBC model keeps it very simple and assumes that a cycle has always a fixed processing time. For timestamping, a clock is used that is always incremented at the end of every cycle by this processing time. This clock is activated with an initial value of 0 seconds when the RBC has received from a train a request to establish a communication session with it. It is deactivated and reset to 0 seconds when the communication session has been terminated.

Transmission of radio messages via GSM-R. GSM-R is out of the scope of the openETCS project and, as a consequence, not formally specified in SCADE. We believe that GSM-R functionalities are located on a lower level and therefore have to be provided by the hardware.

Modeling of the track. Providing a formal specification of the EVC and RBC is only half of the solution. An essential part in the overall context of ETCS plays the track with all its facets such as Eurobalises, signals, speed restrictions and block sections. However, railway operations on a track are impossible without an interlocking. Unfortunately, there exists neither a formal model of an interlocking at the current stage of the openETCS project nor there are any plans for the future to create such one. This circumstance can be explained on the one hand by the fact that the processes within an interlocking are highly complex and depend on national operating rules. What makes it even harder is that there exists no single specification that is accessible to the public. As a consequence, a mechanism is needed that can simulate the behavior of an interlocking as best as possible.

The openETCS project provides a formal track model[7] of the Utrecht - Amsterdam railway line that can be used as a simplified substitute for an interlocking. It serves as a container for predefined track to train messages that are sent to the train during the test run. The underlying concept is shown in Fig. 29.

[7] `http://git.io/vZJHn`, accessed 6 September 2015

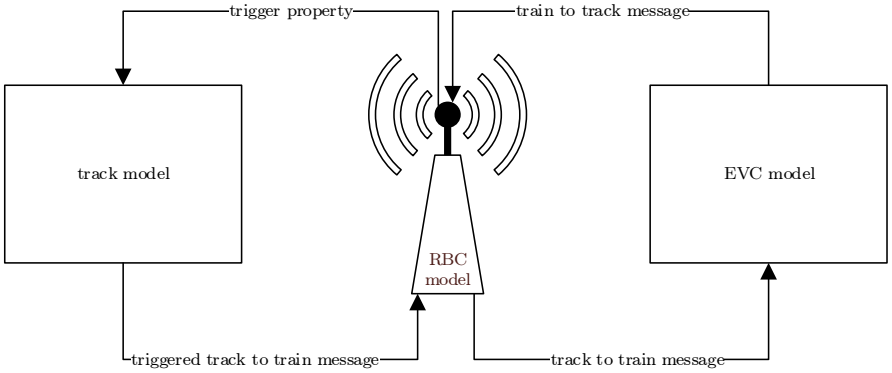


Fig. 29. Concept of the track model

The RBC sends periodically a trigger property to the track model. If the track model finds a track to train message that satisfies this trigger property, then this message, which can be e.g. a movement authority, is triggered and returned to the RBC model for further processing. Put simply, the sending of a trigger property to the track model can be interpreted as a communication process between the RBC and a fictional interlocking.

The trigger property must be well designed. The current track model accepts only an integer value. This is however not sufficient enough for the RBC model because an integer alone is in the overall context meaningless. Therefore, a more complex mechanism is needed. The solution is to analyze the recorded JRU data and to search for possible candidates within its entries that can be used as an appropriate trigger property. A JRU entry is shown below:

```
NID_ENGINE = DB: 1187041
NID_MESSAGE: > M1 <  GENERAL MESSAGE
L_MESSAGE: 70bytes
DATE : Anonymous
TIME: 23:39:36;550
Q_SCALE: 10cm
NID_C: 426, Netherlands, various projects
NID_BG: 602
D_LRBG: 782.4m
Q_DIRLRBG: Nominal
Q_DLRBG: Reverse
L_DOUBTOVER: 173.6m
L_DOUBTUNDER: 86.4m
V_TRAIN: 0km/h s
DRIVER_ID: Anonymous
NID_ENGINE: Anonymous
M_LEVEL: Level STM
M_MODE: Sleeping
```

The solution that was chosen is to use the train position as a trigger property, or more precisely the LRBG (*NID_C* + *NID_BG*) and the distance (*D_LRBG*) in meters to it. To make the track model work properly, the conversion operator *UtrechtAmsterdamScenarioStory00AWrapper* is used in which a position-based trigger property is converted to an integer-based trigger property by executing multiple instances of the operator *RadioTrackTrainMessageTrigger__Trigger* successively (see Fig. 30).

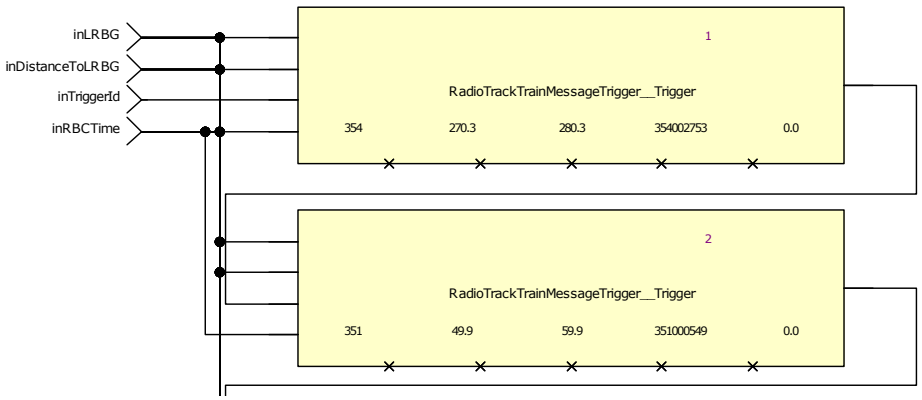


Fig. 30. Converting a position-based trigger property to an integer-based trigger property. A passed LRBG with the identification number 354 and a distance to it that lies within 270.3 m and 280.3 m, would in this case return an integer-based trigger property with the value 354002753.

The operator *RadioTrackTrainMessageTrigger__Trigger* has five hidden inputs. If the passed LRBG equals to the value of the first hidden input and if the distance to it lies within the interval of the second and third hidden input, then the integer value of the fourth hidden input is passed to the track model as an integer-based trigger property. After that, the operator becomes deactivated and is ignored for the rest of the time. The interval check is only performed to compensate possible inaccuracies of a train position report. The fifth hidden input represents a timespan. If it is initialized with a value greater than 0.0, then the operator is armed and the trigger property is only sent if the timespan since it was armed has elapsed. The obligatory requirement for a proper execution in this case is, however, that the LRBG and the distance to it have not changed in meantime. This feature allows a delayed triggering and can be used for more complex scenarios in which a train stands e.g. for a longer period of time in front of a red signal.

Formal specification of the scenarios. The defined scenarios from 5.1 are heavily dependent from each other. To ensure a proper execution of them, a certain workflow has to be strictly adhered by the RBC model (see Fig. 31).

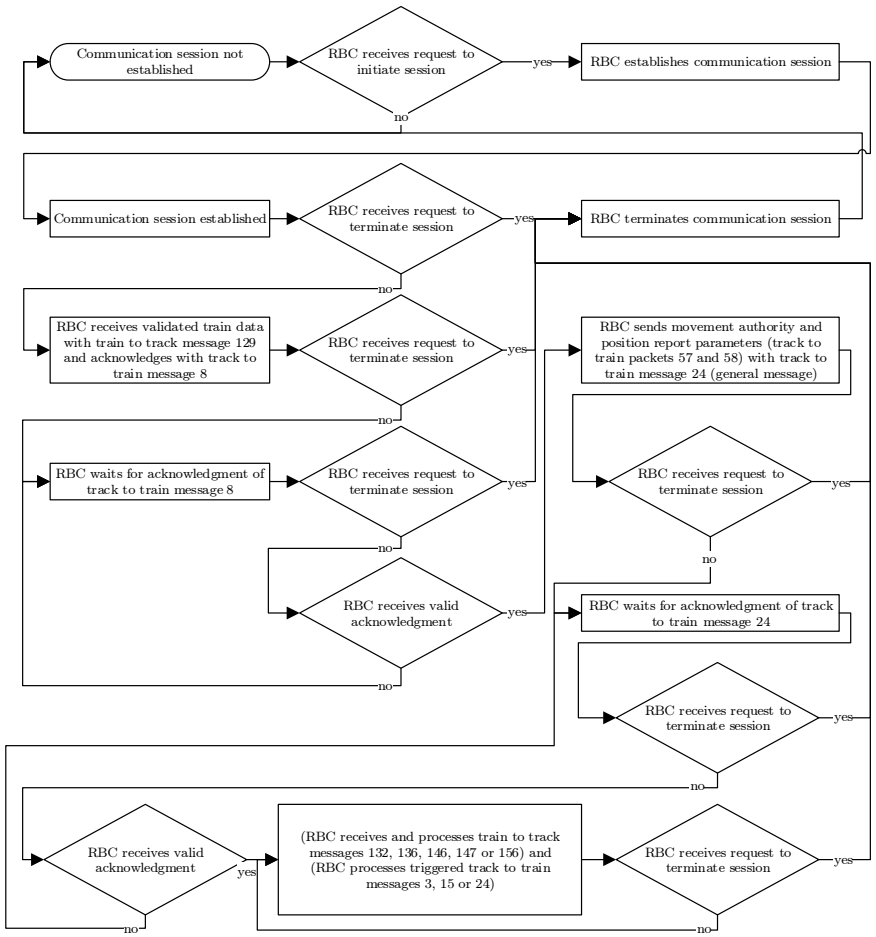


Fig. 31. Workflow of the RBC model

The illustrated workflow is derived from [27, 5.0] and the JRU data. The best way to formally specify such a mechanism in SCADE is to create a controller component that uses state machine constructs. The practical implementation takes place within the operator *RBC__ProcessController*, in which all scenarios are managed. In the following, a description is given how the controller works and how it is formally specified in SCADE. In addition to that, a small case study is performed in which three scenarios are analyzed with respect to their formal specification in SCADE.

Controller. Considering the situation on the track, it is from a simplified point of view always assumed that at the initialization of the RBC model no train is within the operating range of the RBC. This implies that no established communication exists with a train yet. As a consequence, the task of the controller only consists of waiting until the train initiates a communication session with the RBC. This is done within the start state *SESSION_TERMINATED* of the state machine *CONTROLLER_SM*, which is responsible for communication session management related workflow processes such as establishing, maintaining and terminating a communication session (see Fig. 32).

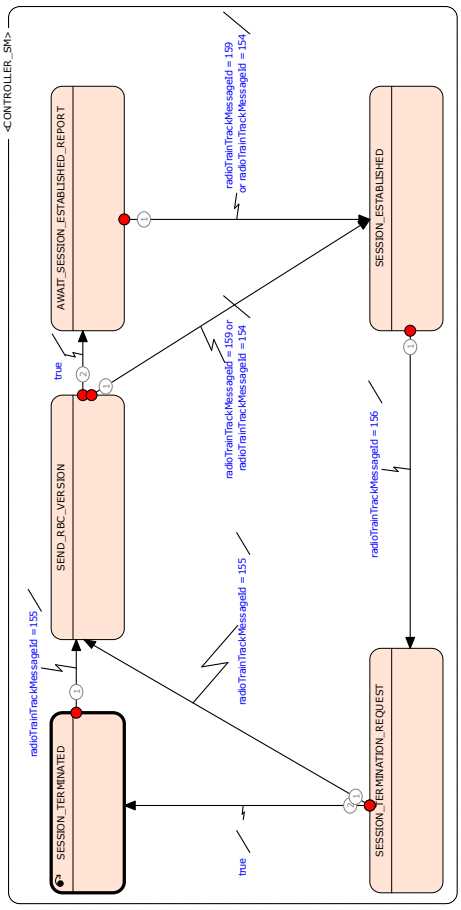


Fig. 32. Controller architecture (1)

Sooner or later, a train enters a piece of track that is within the operating range of the RBC. However, such a train does not automatically initiate a communication session with it. First of all, the train has to be stimulated by an

external event. This is the case when it passes a balise group that contains the packets 42 [27, 7.4.2.10] and 45 [27, 7.4.2.11.1]. This packets order the train to register itself into the radio network of the RBC and to initiate a communication session with it. After a request to initiate a communication session has been received from the train, the controller has to react immediately. This reaction is modeled within the state *SEND_RBC_VERSION* and manifests itself in the fact that the operating version of the RBC is sent to the train with the track to train message 32. Thereby, the version information is read from a configuration source. As a result of this process, the communication session is considered as established for the RBC and the controller is able to maintain it. With regard to the model, the controller switches to the state *SESSION_ESTABLISHED*, which consists of the state machine *SESSION_ESTABLISHED_SM* (see Fig. 33).

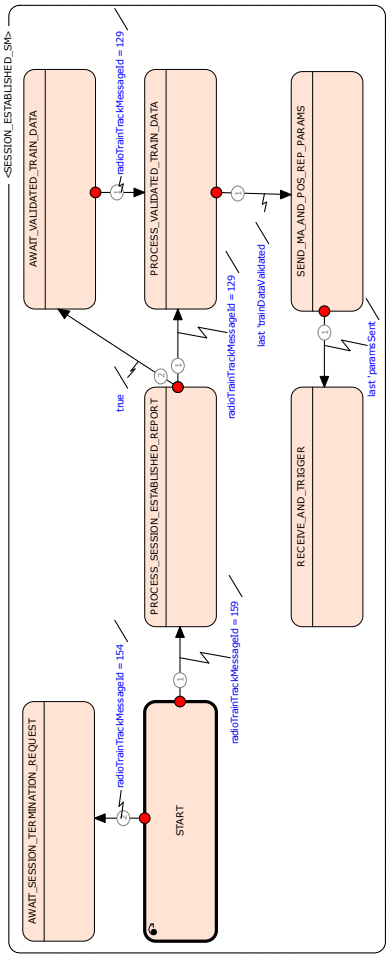


Fig. 33. Controller architecture (2)

However, an established communication session also implies that at every moment a request can be received from the train to terminate it. The sending of such a request always happens when the received operating version of the RBC is incompatible with the version that is supported by the EVC. Also, such a request can be ordered by the RBC itself when it recognizes that the train leaves the operating range. Another possibility is the order from an Eurobalise. The design decision that was chosen was to give the highest priority to the controller regarding the termination of a communication session, i.e., that all workflow processes that are executed within the state *SESSION_ESTABLISHED* are immediately canceled if a request to terminate a communication session has been received. The reason for this decision is that a delayed reaction of the RBC can under some circumstances lead to the situation that a fast moving train has already left the operating range without having received an acknowledgment of this request in time. This means that the communication session has not been terminated in a proper way. As a consequence, the EVC of the train would unnecessarily initiate an emergency brake application. After the cancellation of all processes within the state *SESSION_ESTABLISHED*, a transition is taken to the state *SESSION_TERMINATION_REQUEST* and the RBC acknowledges the request to terminate the communication session instantly with the track to train message 39. If this has been done, then the communication session is considered as terminated for the RBC. The controller is reset to its start state and is able again to process the workflow from the beginning.

According to the workflow, the controller has to ensure that regarding incoming train to track messages, a strict processing sequence is adhered after the communication session has been successfully established. Furthermore, it has to react appropriately if this sequence is violated. Therefore, unexpected train to track messages, except a request to terminate the communication session, are ignored and it is waited until a valid train to track message has been received that is compliant with the workflow.

Considering the processing sequence, the controller has to handle first of all the session establishment report, which is provided with the train to track message 159, or the train to track message 154, which indicates that the train cannot operate with the version of the RBC. However, there is no need to worry about the second case because the openETCS EVC operates with version 3.3.0 and is downward compatible with the operating version of the RBC. Also, such a possible exceptional case is covered within the state *AWAIT_SESSION_TERMINATION_REQUEST*, in which the controller forces the RBC to wait until a session termination request has been received.

If the RBC receives the session established report, then the controller stores all phone numbers of the train that are contained in the optional train to track packet 3 inside the system. This is done within the state *PROCESS_SESSION_ESTABLISHED_REPORT*. Considering the safety aspect, it should be by the way noted that at this stage the exact position of the train is still unknown to the RBC, regardless of how fast the train is moving at the moment. After storing the phone numbers, the controller remains in a waiting state until the validated train data has been received.

The validated train data that is contained in the train to track packet 11 of the received train to track message 129 is extracted and stored inside the system. This happens within the state *PROCESS_VALIDATED_TRAIN_DATA*. Although the storage of the train data is unnecessary work since it is used nowhere in the model for further operations, the key idea behind this step was to provide an easy accessible database that can be used for the implementation of additional useful features in a later iteration step. In addition to the validated train data, a first position report is provided with the train to track packet 0 or 1. This information is used by the controller to update the most recent known train position inside the RBC. Last but not least, the controller ensures that the RBC acknowledges the received train to track message 129 with the track to train message 8, which permits the train to run on the track. After the controller has received a valid acknowledgment of the track to train message 8, the process of receiving the validated train data is considered as finished and the next working step can be executed.

Although the train has already sent a first position report, it still does not know at which time interval it has to periodically report its position to the RBC. This is due to the fact that only the RBC has knowledge about this parameter. Moreover, a complete different aspect also has to be taken into consideration. On the one hand, the RBC can send in certain situations a movement authority to the train without waiting for a request from it. On the other hand, however, the real situation on a track is in many cases quite different from that. Fast moving trains often try to force a new movement authority from the RBC on its own by sending periodically in short time intervals a request to it. This time interval is also unknown to the train. The solution is to transmit the position report and movement authority parameters to the train. This is done by the controller within the state *SEND_MA_AND_POS_REP_PARAMS*. Thereby, the controller reads the parameters from a configuration source and stores them inside the track to train packets 57 and 58 [27, 7.4.2.14-7.4.2.15]. These packets are afterwards merged into the general message 24 that is sent immediately to the train.

The controller continues and enters the state *RECEIVE_AND_TRIGGER* if the train has successfully acknowledged the previously sent general message (see Fig. 34).

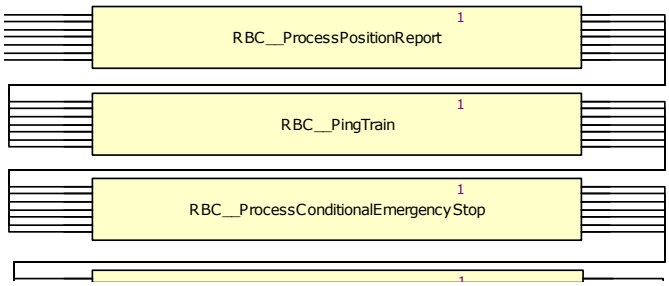


Fig. 34. Controller architecture (3)

According to the workflow, there is no need for the controller to ensure a strict processing sequence anymore. At this stage, the controller is able to manage the following scenarios from 5.1 in parallel:

- Receiving a train position report with the train to track message 136
- Sending a ping to a train (sub-scenario of sending a general message)
- Requesting a train to perform a conditional emergency stop
- Granting a movement authority to a train
- Granting a movement authority request to a train
- Sending a general message to a train (ping excluded)

These scenarios are managed within the state *RECEIVE_AND_TRIGGER* by operators that are all arranged in a daisy chain. On the one hand, this concept ensures a sequential data flow and allows a comfortable concatenation of an arbitrary number of operators, where each of them can be used for the implementation of a certain scenario. As a result, new scenarios can be easily integrated within the RBC model. On the other hand, this mechanism requires some considerations in the arrangement of the operators. Because some of them are dependent from the most recent train position, the operator *RBC__ProcessPositionReport*, which is responsible for receiving the train position report that is contained in the train to track message 136, is arranged at the beginning and therefore executed first by the controller. This ensures that succeeding operators are always aware of the most recent train position.

Case study: receiving a train position report. The RBC has to be always aware of the most recent position of a train. The position is always provided in the form of a position report that is contained in the train to track packet 0 or 1. Both packets can be received in many ways by the RBC. A good example is the train to track message 129 that contains the validated train data as well as a train position report. A reliable mechanism that is used by the train is the periodical sending of the train to track message 136 according to the position report parameters of the RBC. This kind of scenario is formally specified within the operator *RBC__ProcessPositionReport* (see Fig. 35).

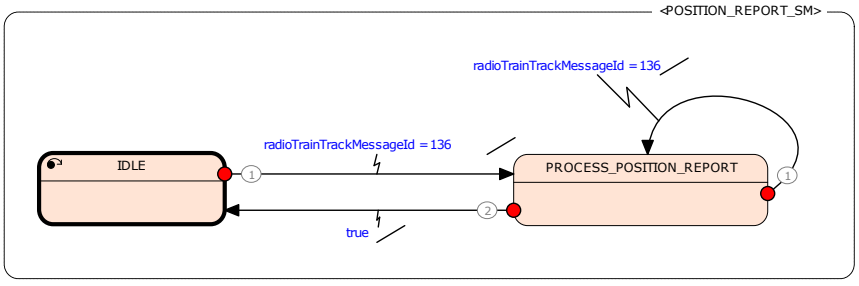


Fig. 35. Receiving a train position report

If a train position report has been received with the train to track message 136, then a strong transition is taken from the start state *IDLE* to the state *PROCESS_POSITION_REPORT*. Within this state, the train position report is processed (see Fig. 36).

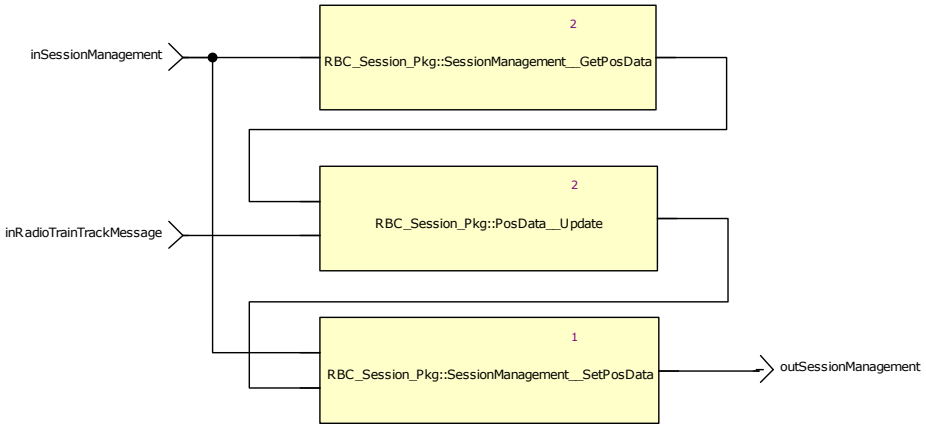


Fig. 36. Updating the train position

The train position is extracted from the train to track packet 0 or 1 and stored inside the system by applying the operators *PosData__Update* and *SessionManagement__SetPosData* successively. A strong reflexive transition is taken afterwards if the train to track message 136 is received in the following cycle again. Otherwise, the state *IDLE* is reached.

In conclusion, it must be pointed out that no safety checks are performed by the RBC regarding its position report parameters. If the periodically sent train to track message 136 is not received in time, then no reaction takes place.

Case study: sending a ping. A ping is always sent as a general message that has not to be acknowledged. Considering safety aspects, it plays an important role because it indicates to a train that the communication session is still established.

There exist different manufacturer specific solutions. Some implementations send periodically a ping within a fixed time interval. Other implementations send for reasons of redundancy two pings at once. In principle, however, there is no right or wrong. It is only important that the train receives a consistent track to train message within its fixed contact time, independently of whether it is a ping or not.

An efficient approach to send a ping message is realized within the operator *RBC__PingTrain*. This operator checks first of all if certain criteria are met that make it necessary to send a ping (see Fig. 37).

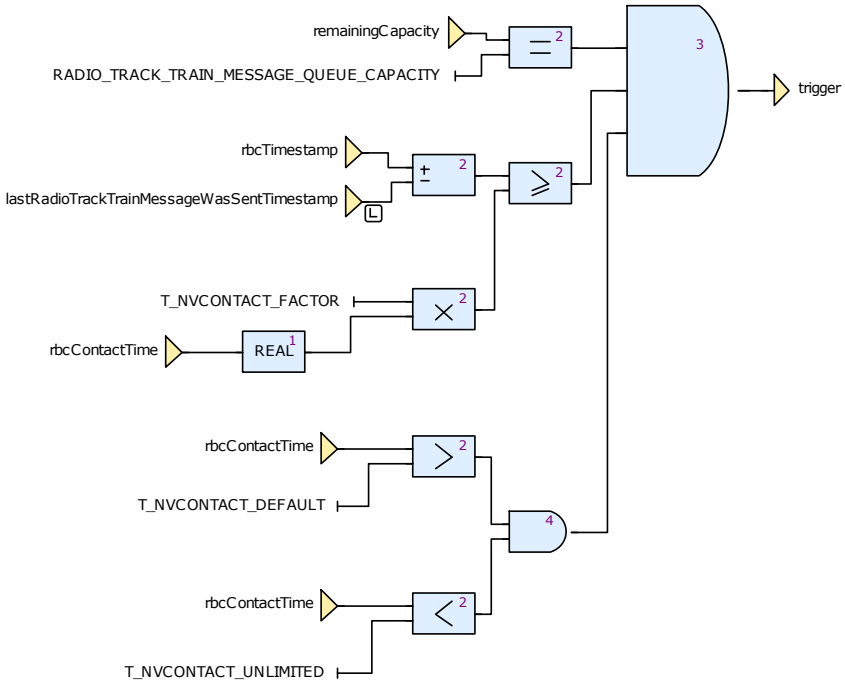


Fig. 37. Condition for sending a ping

It is checked if the difference of the current value of the timestamp clock, represented by the local variable *rbcTimestamp*, and the timestamp of the last sent track to train message, represented by the local variable *lastRadioTrackTrainMessageWasSentTimestamp*, is greater or equal than a given threshold. This threshold results from the multiplication of a safety factor *T_NVCONTACT_FACTOR* and the contact time, which is represented by the local variable *rbcContactTime*. Because the transport over GSM-R takes some time, the value of the constant *T_NVCONTACT_FACTOR* is set to 0.75. This ensures that the threshold is always smaller than the required contact time.

If the threshold condition becomes true, then it would be on the one hand a legitimate design decision to create immediately a ping message and to buffer it. On the other hand, this would waste memory within the buffer if it already contains at least one track to train message. Because a track to train message is always taken away from the buffer at the end of a cycle and sent immediately to the train, the train would receive a track to train message anyway within the required contact time. So, it would be in that case not necessary to send a ping. As a consequence, it is additionally checked if the buffer is empty. The result of the threshold and capacity check is written into the local boolean variable *trigger*. If this variable takes the value *true*, then a ping message is created and afterwards buffered.

Case study: granting a movement authority and a movement authority request. The scenarios of granting a movement authority and a movement authority request are both formally specified within the operator *RBC__ProcessMovementAuthority* (see Fig. 38).

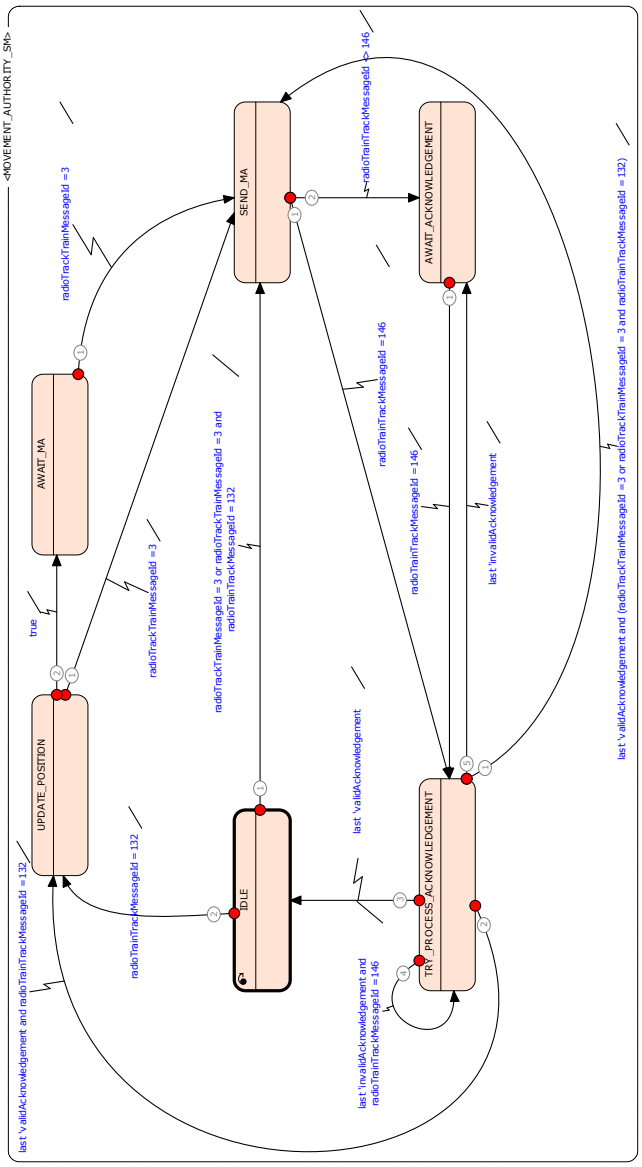


Fig. 38. Granting a movement authority and a movement authority request

If a movement authority has been triggered by the track model alone or simultaneously with a received movement authority request, then the state *SEND_MA* is reached from the start state *IDLE*. Within this state, the triggered movement authority message is buffered and the train position is updated in the case that the movement authority has been triggered simultaneously with a movement authority request. To put in a nutshell, this reaction can be interpreted as a decision of a fictional interlocking that allows the RBC to grant a movement authority to a train on its own initiative or due to a received movement authority request.

If only a movement authority request has been received by the RBC, then the state *UPDATE_POSITION* is reached. Within this state, the train position report that is contained in the train to track packet 0 or 1 of the received movement authority request is extracted and stored inside the system. Before the state *SEND_MA* can be reached afterwards, it is waited until a movement authority has been triggered by the track model. Simply speaking, the RBC waits for an answer from a fictional interlocking.

Every triggered movement authority has to be acknowledged with the train to track message 146. Thereby, a received acknowledgment is checked within the state *TRY_PROCESS_ACKNOWLEDGEMENT*. First of all, the value of the message identification number of the previous sent track to train message is get by the operator *RadioTrackTrainHeader__Get_NID_MESSAGE* and compared to the value 3 (see Fig. 39). If both values are equal, then the previous sent track to train message was a movement authority. If this is the case, then its timestamp is assigned to the local variable *radioTrackTrainMessageWasSentTimestamp*. Otherwise, this variable holds its last default value 0.0.

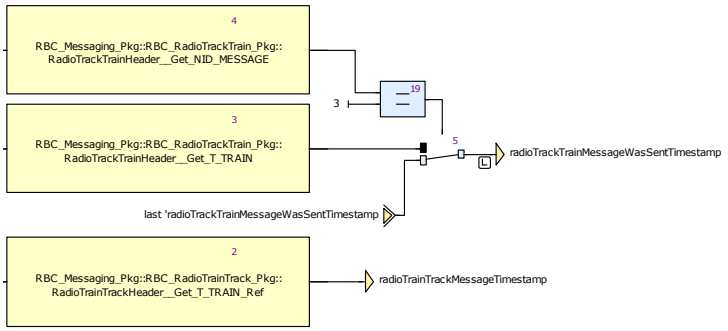


Fig. 39. Acknowledgment processing of a train to track message (1)

If the reference timestamp of the received acknowledgment, which is represented by the local variable *radioTrainTrackMessageTimestamp*, is equal to the value of the variable *radioTrackTrainMessageWasSentTimestamp*, then the acknowledgment is valid. The signal *validAcknowledgement* is emitted and the RBC is able again to process a new movement authority and a new movement

authority request (see Fig. 40). If the acknowledgment is invalid, then the signal *invalidAcknowledgement* is emitted. The RBC must then wait until another acknowledgment has been received and perform a validity check on it again.

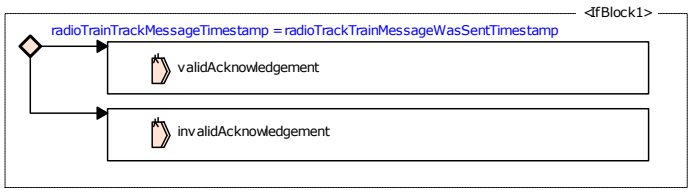


Fig. 40. Acknowledgment processing of a train to track message (2)

In conclusion, it should be mentioned that the described mechanisms are kept very simple. Safety checks regarding the movement authority parameters of the RBC are not performed. Furthermore, the train and RBC can be under some circumstances affected in a large degree. If the state *AWAIT_MA* is active, then the control flow can only continue if the track model triggers a movement authority. However, this may cause a deadlock-like situation if no movement authority is triggered for a long time. If this case occurs, then the train will not receive a movement authority in time. As a consequence, the EVC would take appropriate safety measures and possibly initiate an emergency brake application. The repeated transmission of a movement authority request would also have no effect because the state *AWAIT_MA* would be due to its outgoing transition condition still active. A possible solution would be e.g. to add a timeout for the triggering of a movement authority. If this timeout exceeds, then a transition to another safe state could be taken.

5.3 Integration.

To obtain a working model of the RBC, all relevant components have to be integrated. First of all, components, such as the buffer, clock, consistency checker and controller, are brought together within the operator *RBC*. Furthermore, it should not be forgotten that the track plays also an important role for the RBC model. As a consequence, the external track model of the Utrecht - Amsterdam railway line is connected with the RBC model. This takes place within the operator *RBCIntegration_Utrecht_Amsterdam* (see Fig. 41).

The converter *UtrechtAmsterdamScenarioStory00AWrapper* gets from the RBC model the most recent train position and calculates from it an integer-based trigger property that is passed to the track model *Story00A_RBC*. Depending on the value of this trigger-property, a track to train message is triggered and fed back to the RBC model, in which it is processed by the controller. Because of the unavoidable feedback loop between the RBC and track model, a followed by operator *FBY* must be used that delays the triggering for one cycle.

The separation of the track model and its corresponding converter from the RBC model results in a loosely coupled architecture. This allows a replacement of the track at any time without affecting the RBC model.

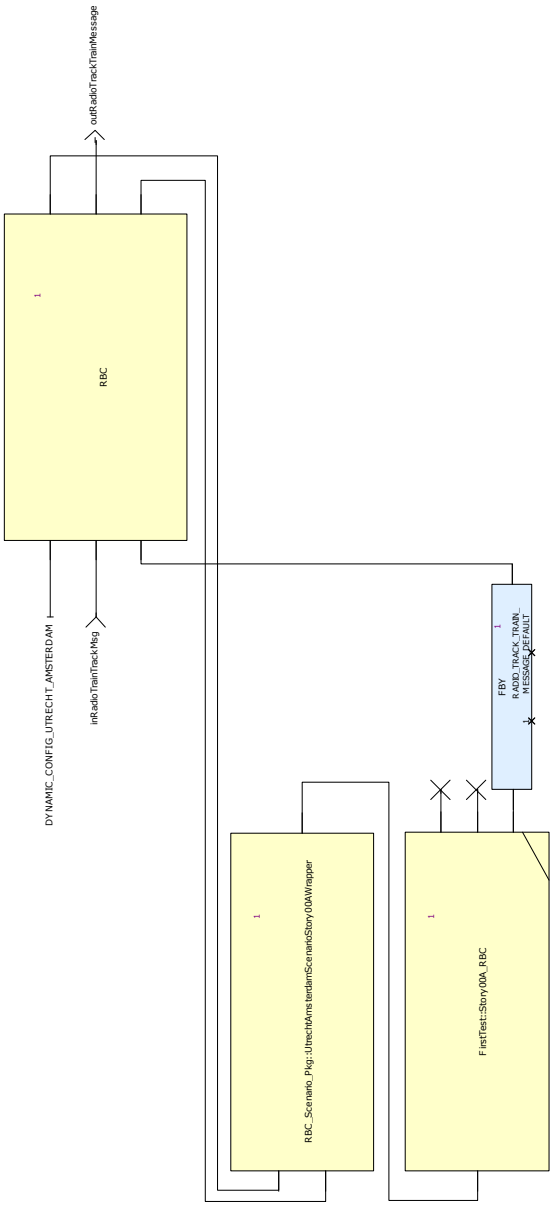


Fig. 41. Integration of the RBC and track model

6 Test of the RBC model

By testing the scenarios from 5.1 against the RBC model, it is possible to find critical errors in the formal specification. A global test environment already exists as an external SCADE project[8]. It provides a graphical user interface, including a DMI, that allows to perform in the SCADE simulator a test run on the Utrecht - Amsterdam track (see Fig. 42).

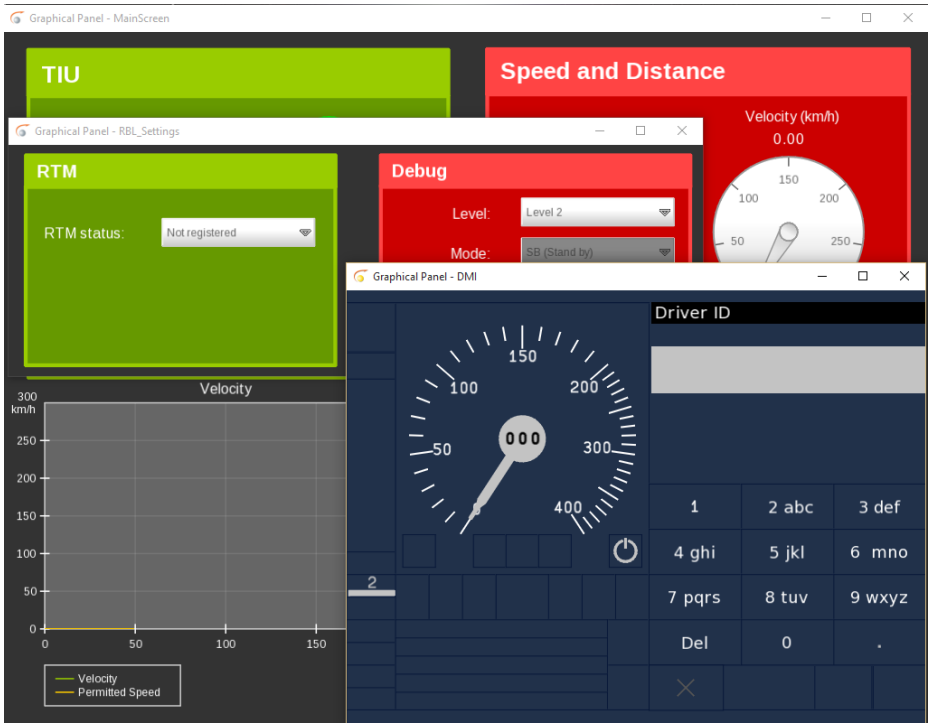


Fig. 42. Graphical test environment

The test environment is heavily dependent from the EVC model, which is in a constant development process. Unfortunately, inconsistencies in the EVC model produce at the current stage compilation errors in the test environment so that it is impossible to use it for simulation purposes. However, it was in the last functioning version possible to reach the controller state *RECEIVE_AND_TRIGGER* and to receive train position reports with the train to track message 136 (see Fig. 34 and 35). Thereby, no errors could be found in the workflow of the RBC model. Furthermore, all track to train messages were sent correctly.

[8] http://git.io/vnQte, accessed 24 September 2015

The only thing that can be done at this moment is to isolate the RBC model from the EVC model and simulate it alone. However, this impedes the simulation process and results in a time-consuming work because input variables that are normally automatically provided by the EVC model must be set manually cycle by cycle with pseudo values (see Fig. 43).

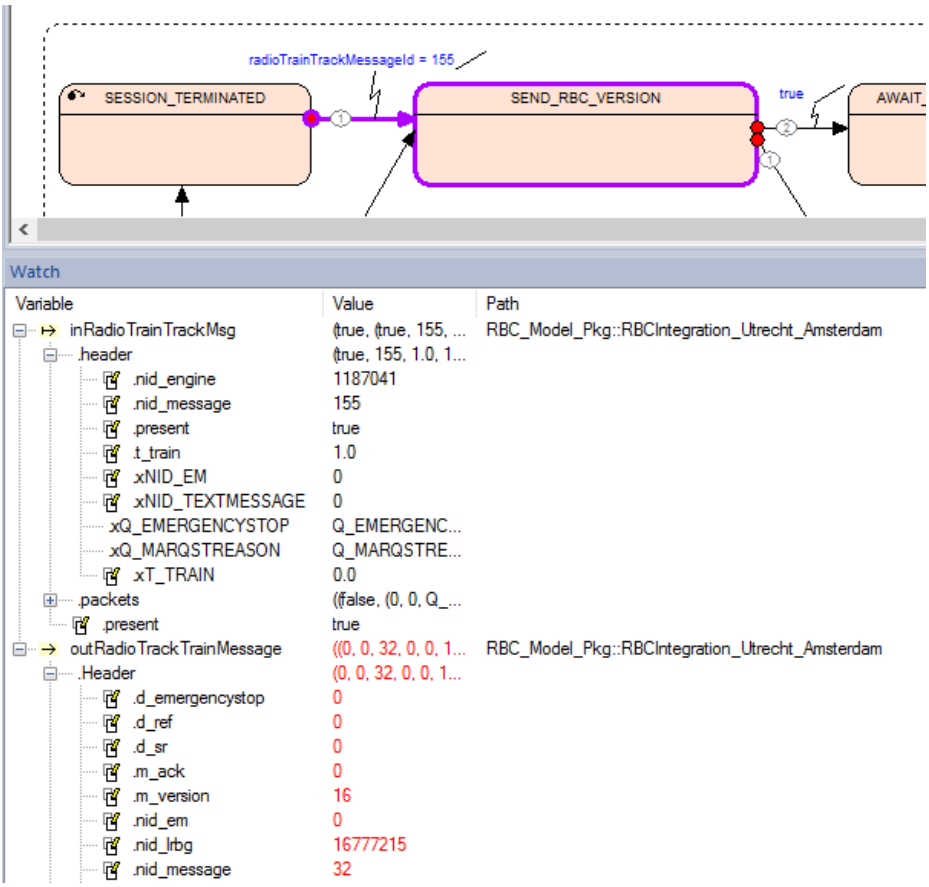


Fig. 43. Manual simulation of the RBC model

During the simulation, only one critical error was detected that concerns the termination of an established communication session by the RBC controller (see Fig. 32). Looking back to 5.1 again, a communication session is considered as terminated for the RBC if it has sent the track to train message 39. Thereby, it should not be forgotten that all track to train messages are stored inside a FIFO buffer within the RBC model. It is now possible that older elements still remain inside this buffer and that the track to train message 39 is enqueued at the last free

position of it. According to the FIFO principle, it is sent last. In this constellation, the communication session is still active but a strong priority based transition is illegally taken from the state *SESSION_TERMINATION_REQUEST* to the state *SESSION_TERMINATED* or *SEND_RBC_VERSION*. To fix this bug, the buffer could be cleared within the state *SESSION_TERMINATION_REQUEST* to add afterwards immediately the track to train message 39 to it (see Fig. 44). This would ensure that the buffer contains only this single track to train message that is sent at the end of the same cycle to the train.

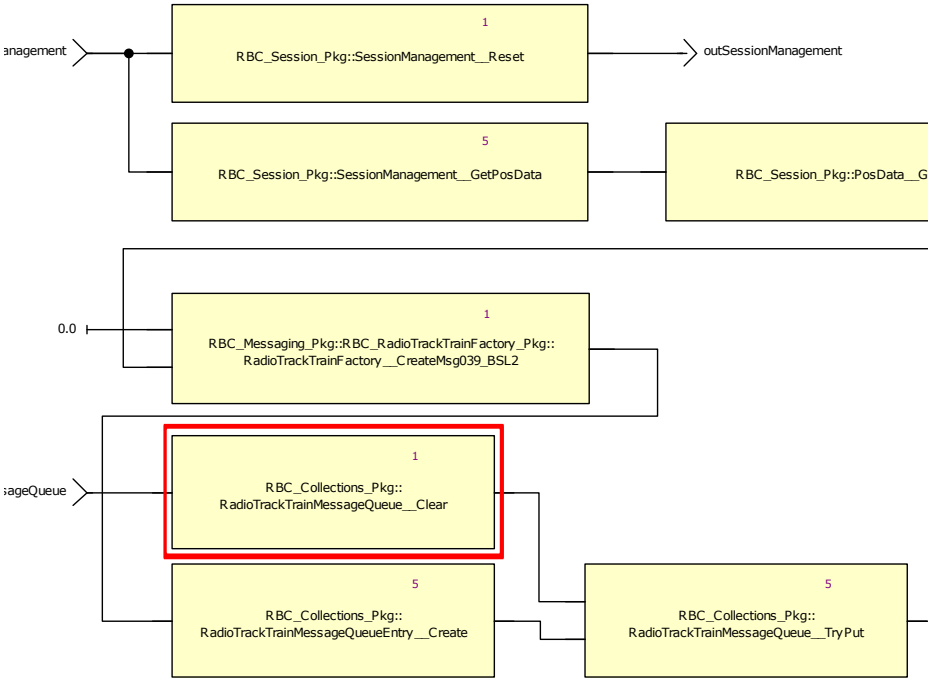


Fig. 44. Correction of the RBC model. The operator *RadioTrackTrainMessageQueue__Clear* (marked in red) clears the buffer. Afterwards, the track to train message 39 is added to it by applying the operator *RadioTrackTrainMessageQueue__TryPut*.

Despite of the intensive simulation that was performed, it must be emphasized that no statement can be made about the correctness of the formal specification without having it formally verified. However, a formal verification of the RBC model is due to a decision of the responsible project members currently out of the scope of the openETCS project.

7 Conclusion

Formal methods have proven their usefulness in industry over years. This has been in particular demonstrated by a large number of successful industrial projects in the field of railway signaling systems. With the RBC, a safety-critical system was introduced, whose behavior was modeled by means of a formal specification.

After the analysis of requirements, a formal specification of the RBC was created in SCADE. Afterwards, a test of the resulting RBC model was performed in form of a simulation. Thereby, quite positive results could be achieved. This was, in particular, reflected in the number of detected errors. During the simulation, only one critical error could be found. Moreover, this error could be solved very quickly.

The SCADE environment guided successfully through the development process. The provided simulator and model checker made it possible to detect errors in early phases of the development and allowed, based on the given feedback, a stepwise improvement of the formal specification. This led to a, apart from a few exceptions, SRS-compliant and unambiguous formal specification of the RBC. However, it must be also pointed out that SCADE is no miracle cure. In contrast to the established high-level programming languages, many basic operations, such as the use of loop constructs, require a change in the mindset and sometimes a lot of effort. Furthermore, no real design patterns exist and therefore it is difficult to plan the system architecture in advance. A lack of discipline can lead rapidly to a formal specification that is hard to understand and difficult to maintain.

It has to be mentioned that the RBC model is kept very simple at this stage of development. Strictly speaking, it is rather a mock-object that simulates the inputs for the EVC. To make the behavior of the RBC more realistic, it would be from my point of view useful to extend the model with further features that could not be implemented within this master's thesis due to a lack of time. This includes, amongst others, the monitoring of timing constraints.

On the whole, it can be said that the resulting system is a success. It could be demonstrated that the application of formal methods has positive effects on the safety and quality of a software system. To conclude, the formal RBC model makes a contribution towards interoperability in the European railway traffic and helps to achieve the ambitious goals of the openETCS project.

List of Figures

List of Tables

References

1. Abrial, J.R.: A Formal Approach To Large Software Construction. In: van de Snepscheut, J.L.A. (ed.) MPC. Lecture Notes in Computer Science, vol. 375, pp. 1–20. Springer (1989)
2. Accellera Systems Initiative: Property Specification Language Reference Manual. `http://www.eda.org/vfv/docs/psl_lrm-1.01.pdf` (2003), accessed 26 Apr 2015
3. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
4. Barroca, L.M., McDermid, J.A.: Formal Methods: Use and Relevance for the Development of Safety-Critical Systems. Computer Journal 35(6), 579–599 (1992)
5. Behm, P., Benoit, P., Faivre, A., Meynadier, J.M.: Météor: A Successful Application of B in a Large Project. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) World Congress on Formal Methods. Lecture Notes in Computer Science, vol. 1708, pp. 369–387. Springer (1999)
6. Bidoit, M., Gaudel, M.C., Mauboussin, A.: How to Make Algebraic Specifications More Understandable: An Experiment with the PLUSS Specification Language. Sci. Comput. Program. 12(1), 1–38 (1989)
7. Bjørner, D.: New results and trends in formal techniques and tools for the development of software for transportation systems - A review. In: Proceedings 4th Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'03). L'Harmattan Hongrie, Budapest (2003)
8. Bloem, R., Cimatti, A., Greimel, K., Hofferek, G., Könighofer, R., Roveri, M., Schuppan, V., Seeber, R.: RATSY - A New Requirements Analysis Tool with Synthesis. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV. Lecture Notes in Computer Science, vol. 6174, pp. 425–429. Springer (2010)
9. Bowen, J.P., Stavridou, V.: Safety-critical systems, formal methods and standards. Software Engineering Journal 8(4), 189–209 (Jul 1993)
10. Caspi, P., Pilaud, D., Halbwachs, N., Plaice, J.: Lustre: A Declarative Language for Programming Synchronous Systems. In: POPL. pp. 178–188. ACM Press (1987)
11. Chiappini, A., Cimatti, A., Porzia, C., Rotondo, G., Sebastiani, R., Traverso, P., Villafiorita, A.: Formal Specification and Development of a Safety-Critical Train Management System. In: Felici, M., Kanoun, K., Pasquini, A. (eds.) SAFECOMP. Lecture Notes in Computer Science, vol. 1698, pp. 410–419. Springer (1999)
12. Coq: `https://coq.inria.fr`, accessed 31 May 2015
13. Counsell, S., Núñez, M.: Software Engineering and Formal Methods - SEFM 2013 Collocated Workshops: BEAT2, WS-FMDS, FM-RAIL-Bok, MoKMaSD, and OpenCert, Madrid, Spain, September 23-24, 2013, Revised Selected Papers. Springer, Berlin, Heidelberg (2014)
14. D'Argenio, P.R., Hermanns, H., Katoen, J.P., Klaren, R.: MoDeST - A Modelling and Description Language for Stochastic Timed Systems. In: de Alfaro, L., Gilmore, S. (eds.) PAPM-PROBMIV. Lecture Notes in Computer Science, vol. 2165, pp. 87–104. Springer (2001)
15. Department of Defense: MIL-STD-882D: Standard practice for system safety (Jan 1993)
16. Dormoy, X.: SCADE 6 - A Model Based Solution For Safety Critical Software Development
17. Dowson, M.: The Ariane 5 software failure. SIGSOFT Software Engineering Notes 22(2), 84 (1997)
18. Esterel Technologies: `http://www.esterel-technologies.com`, accessed 27 September 2015

19. European Commission: Council directive 96/48/EC of 23 July 1996 on the interoperability of the trans-European high-speed rail system (Jul 1996)
20. European Commission: Directive 2001/16/EC of the European Parliament and the council of 19 March 2001 on the interoperability of the trans-European conventional rail system (Mar 2001)
21. European Commission: Trans-European Transport Network: Ten-T Priority Axes and Projects 2005 (2005)
22. European Commission: Directive 2008/57/EC of the European Parliament and the council of 17 June 2008 on the interoperability of the rail system within the Community (Jun 2008)
23. European Commission: EU transport in figures (2014)
24. European Committee for Electrotechnical Standardization: EN 50128: Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems (Jun 2011)
25. European Railway Agency: Memorandum of Understanding concerning the strengthening of cooperation for the management of ERTMS (Apr 2012)
26. European Railway Agency: New Annex A for ETCS Baseline 3 and GSM-R Baseline 0 (Apr 2012), accessed 21 June 2015
27. European Railway Agency: System Requirements Specification Subset 026-3.3.0 (Mar 2012)
28. European Railway Agency: ETCS Driver Machine Interface 3.4.0 (May 2014)
29. Fang, H., Shi, J., Zhu, H., Guo, J., Larsen, K.G., David, A.: Formal Verification and Simulation for Platform Screen Doors and Collision Avoidance in Subway Control Systems. International Journal on Software Tools for Technology Transfer 16(4), 339–361 (Aug 2014)
30. Ferrari, A., Fantechi, A., Magnani, G., Grasso, D., Tempestini, M.: The Metrô Rio case study. Science of Computer Programming 78(7), 828–842 (2013)
31. Galton, A.: Temporal logics and their applications. Academic Press, London (1987)
32. George, C., Haxthausen, A.E.: The Logic of the RAISE Specification Language. In: Logics of Specification Languages, pp. 349–399. Springer (2008)
33. Gjaldbæk, T., Haxthausen, A.E.: Modelling and Verification of Interlocking Systems for Railway Lines. In: Proceedings of the 10th IFAC Symposium on Control in Transportation Systems (2003)
34. Goguen, J., Tardo, J.: An Introduction to Obj: a Language for Writing and Testing Formal Algebraic Program Specifications (1979)
35. Harel, D.: Statecharts: a visual formalism for complex systems. Science of Computer Programming 8(3), 231–274 (1987)
36. Harel, D., Naamad, A.: The statemate semantics of statecharts. ACM Transactions on Software Engineering and Methodology 5(4), 293–333 (October 1996)
37. Hase, K.R.: OpenETCS soll Zugsicherung interoperabel, sicher und bezahlbar machen. SafeTRANS News 2, 3 (2012)
38. Hase, K.R.: Market Relevance for openETCS (Jul 2013), `http://git.io/vc8YM`, accessed 27 September 2015
39. Haxthausen, A.E.: An Introduction to Formal Methods for the Development of Safety-critical Applications. `http://www2.imm.dtu.dk/~aeha/Railway/index/TSnotat.pdf` (Aug 2010), accessed 14 May 2015
40. Haxthausen, A.E., Bliguet, M.L., Kjær, A.A.: Modelling and Verification of Relay Interlocking Systems. In: Choppy, C., Sokolsky, O. (eds.) Monterey Workshop. Lecture Notes in Computer Science, vol. 6028, pp. 141–153. Springer (2008)

41. Haxthausen, A.E., Peleska, J.: Formal Development and Verification of a Distributed Railway Control System. IEEE Transactions on Software Engineering 26(8), 687–701 (2000)
42. Haxthausen, A.E., Peleska, J., Kinder, S.: A formal approach for the construction and verification of railway control systems. Formal Aspects of Computing 23(2), 191–219 (2011)
43. Hennebert, C., Guiho, G.D.: SACEM: A Fault Tolerant System for Train Speed Control. In: FTCS. pp. 624–628. IEEE Computer Society (1993)
44. Henzinger, T.A.: The theory of hybrid automata. In: Logic In Computer Science. pp. 278–292. IEEE Computer Society Press (1996)
45. Hermanns, H., Jansen, D.N., Usenko, Y.S.: From StoCharts to MoDeST: a comparative reliability analysis of train radio communications. In: WOSP '05. pp. 13–23. ACM (2005)
46. Hoare, C.A.R.: An Axiomatic Basis for Computer Programming. Communications of the ACM 12(10), 576–580, 583 (October 1969)
47. Hyun-Jeong, J., Jong-Gyu, H., Yong-Ki, Y.: Development of Formal Method Application for Ensuring Safety in Train Control System. `http://www.railway-research.org/IMG/pdf/o.3.4.2.3.pdf` (2008)
48. International Organization for Standardization, International Electrotechnical Commission: ISO/IEC 15408: Common Criteria for Information Technology Security Evaluation
49. ITU: Z.100: Specification and Description Language - Overview of SDL-2010. `http://www.itu.int/rec/T-REC-Z.100`, accessed 26 Apr 2015
50. ITU: Z.120: Message Sequence Chart (MSC). `http://www.itu.int/rec/T-REC-Z.120`, accessed 26 Apr 2015
51. Jacky, J.: The way of Z: practical programming with formal methods. Cambridge University Press (1996)
52. Jones, C.: Systematic Software Development Using VDM. International Series in Computer Science, Prentice Hall, 2nd edn. (1991)
53. Kröger, F.: Temporal Logic of Programs, EATCS Monographs on Theoretical Computer Science, vol. 8. Springer (1987)
54. Lakehal, A., Parissis, I.: Structural Test Coverage Criteria for Lustre Programs. In: Proceedings of the 10th International Workshop on Formal Methods for Industrial Critical Systems. pp. 35–43. FMICS '05, ACM, New York, NY, USA (2005)
55. Laurent, O., Michel, P., Wiels, V.: Using Formal Verification Techniques to Reduce Simulation and Test Effort. In: Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity. pp. 465–477. FME '01, Springer-Verlag, London, UK, UK (2001)
56. Lecomte, T., Servat, T., Pouzancre, G.: Formal Methods in Safety-Critical Railway Systems. 10th Brasilian Symposium on Formal Methods (2007)
57. Leveson, N., Turner, C.: An investigation of the Therac-25 accidents. Computer 26(7), 18–41 (1993)
58. Lin, Z., Tao, T., Ruijun, C., Liyun, H.: Property Based Requirements Analysis for Train Control System. Journal of Computational Information Systems 9(3), 915–922 (2013)
59. Lindegaard, M.P., Viuf, P., Haxthausen, A.E.: Modelling Railway Interlocking Systems. In: Proceedings of the 9th IFAC Symposium on Control in Transportation Systems. pp. 211–217 (Jun 2000)
60. MathWorks: Stateflow® User's Guide. `http://www.mathworks.com/help/pdf_doc/stateflow/sf_ug.pdf`, accessed 26 Apr 2015

61. Max Planck Institut: SPASS, http://www.spass-prover.org, accessed 31 May 2015
62. MetaMath: http://us.metamath.org/mpegif/mmset.html, accessed 31 May 2015
63. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc. (1989)
64. Mizar: http://www.mizar.org, accessed 31 May 2015
65. Myers, W.: Can Software for the Strategic Defense Initiative Ever Be Error-Free? Computer 19(11), 61–67 (Nov 1986)
66. Open Proofs: http://www.openproofs.org/wiki/Main_Page, accessed 27 September 2015
67. openETCS: ITEA Co-summit 2015 (2015), http://git.io/vctbb, accessed 27 September 2015
68. Pénin, J., Hussler, C., Burger-Helmchen, T.: New shapes and new stakes: a portrait of open innovation as a promising phenomenon. Journal of Innovation Economics 11(7) (Jan 2011)
69. Peterson, J.L.: Petri Nets. ACM Computing Surveys 9(3), 223–252 (1977)
70. Petite-Doche, M., Matthias, G.: OpenETCS process. Definition of the overall process for the formal description of ETCS and the rail system it works in. Tech. Rep. D2.3, openETCS (Jun 2013), http://git.io/vLsHL, accessed 16 June 2015
71. Pixabay: https://pixabay.com/de/signal-stopp-s-bahn-gleisanlagen-282880, accessed 23 September 2015
72. PVS: http://pvs.csl.sri.com, accessed 31 May 2015
73. Radio Technical Commission for Aeronautics: DO-178C: Software Considerations in Airborne Systems and Equipment Certification
74. Rajan, S., Shankar, N., Srivas, M.K.: An Integration of Model Checking with Automated Proof Checking. In: Wolper, P. (ed.) CAV. Lecture Notes in Computer Science, vol. 939, pp. 84–97. Springer (1995)
75. Schweizerische Eidgenossenschaft: Schlussbericht der Unfalluntersuchungsstelle Bahnen und Schiffe über die Entgleisung von Güterzug 43647 der BLS AG auf der Weiche 34 (Einfahrt Lötschberg-Basisstrecke) (Jun 2008), http://www.sust.admin.ch/pdfs/BS//pdf/07101601_SB.pdf, accessed 27 September 2015
76. Siemens Mobility: http://www.siemens.com/press/photo/SOIMO201110-03d, accessed 23 September 2015
77. Software Engineering Standards Committee of the IEEE Computer Society: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (2000)
78. Standish Group: Chaos Report (1994)
79. Stanley, P.: ETCS for Engineers. Eurail Press (2011)
80. TU München: Isabelle, http://isabelle.in.tum.de, accessed 31 May 2015
81. UK Ministry of Defence: Def Stan 00-55: Requirements for Safety Related Software in Defence Equipment (Aug 1997)
82. UK Ministry of Defence: Def Stan 00-56: Safety Management Requirements for Defence Systems (Jun 2007)
83. University of Augsburg: KIV, http://www.isse.uni-augsburg.de/software/kiv, accessed 31 May 2015
84. University of New Mexico: Prover9, https://www.cs.unm.edu/~mccune/prover9, accessed 31 May 2015
85. University of Texas at Austin: ACL2, http://www.cs.utexas.edu/users/moore/acl2, accessed 31 May 2015
86. Voss, K.: Using Predicate/Transition-Nets to Model and Analyze Distributed Database Systems. IEEE Transactions on Software Engineering 6(6), 539–544 (1980)

87. Winter, P.: ERTMS Training Programme 2011, Session B2 (2011)
88. Woodcock, J., Larsen, P.G., Bicarregui, J., Fitzgerald, J.S.: Formal methods: Practice and experience. ACM Computing Surveys 41(4) (2009)

A Digital Edition

B Overview of relevant packets and radio messages

B.1 Track to train packets.

ID	Name	Description
42	Session management	Packet to give the identity and telephone number of the RBC with which a session shall be established or terminated.
45	Radio network registration	Packet to give the identity of the Radio Network to which a registration shall be enforced.
57	Movement authority request parameters	This packet is intended to give parameters telling when and how often the train has to ask for a movement authority.
58	Position report parameters	This packet is intended to give parameters telling when and how often the position has to be reported.

B.2 Train to track packets.

ID	Name	Description
0	Position report	This packet is used to report the train position and speed as well as some additional information (e.g. mode, level, etc.)
1	Position report based on two balise groups	This packet is an extension of the “standard position report“ packet 0. It is used in case of single balise groups if the orientation of the LRBG is unknown but the on-board equipment is able to report a second balise group (the one detected before) to give a direction reference for the directional information in the position report.
3	On-board telephone numbers	Telephone numbers associated to the on-board equipment
11	Validated train data	Validated train data.

B.3 Track to train messages.

ID	Name
3	Movement authority
8	Acknowledgment of train data
15	Unconditional emergency stop
24	General message
32	RBC operating version
39	Acknowledgment of termination of a communication session

B.4 Train to track messages.

ID	Name
129	Validated train data
132	Train position report
136	Movement authority request
146	Acknowledgment
147	Acknowledgment of emergency stop
154	No compatible version supported
155	Initiation of a communication session
156	Termination of a communication session
159	Session established

C Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht, sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

Ort, Datum Unterschrift