

Using Formal Methods for Verification and Validation in Railway

Klaus Reichl, Tomas Fischer, and Peter Tummeltshammer^(✉)

Thales Austria GmbH, Handelskai 92, 1200 Vienna, Austria
{klaus.reichl,tomas.fischer,peter.tummeltshammer}@thalesgroup.com

Abstract. A very promising and efficient method of showing the correctness of a complex system is using formal methods on a model of that system. To this end there exist plentiful methods and tools for easing the mathematically burdensome process of refinement and proofs, as well as the computationally complex task of model checking.

While in todays industrial applications formal methods are mostly used for verification (i.e. for showing that the system model fulfills properties such as completeness and consistency) we propose to use these methods for validation as well (i.e. correspondence of the model with the customer needs).

In this paper we show the applicability as well as the limitations of this approach for feature driven development towards continuous verification and validation. As an example we present a model of a railway interlocking system written in Event-B.

The model can be instantiated and animated, which in combination with model checking and formal proofs demonstrates the usefulness of the approach.

The resulting model can be used again to automatically generate test cases which are suitable to show the correspondence of the implementation and the model, given that the model supports a sufficient level of detail.

Keywords: Formal methods · Event-B · Verification · Validation

1 Introduction

The railway domain is characterized by high safety as well as reliability and availability requirements which demand thorough verification and validation. The long lifespan (25+ years) of railway systems induces need for changes caused by feature enhancements or subsystem renewal. No modification may compromise the required safety properties. This implies the necessity to reason about the system (and thus about the model) not only in the development phase, but during the whole product lifespan. The ambition to keep the cost of change minimal demands incremental verification and validation instead of full verification and validation of the whole system.

However, today’s systems are too complex to be reasoned about by experts only. It is difficult to comprehend all implications of a modification, some unintended side effects can be overlooked very easily and may have fatal consequences.

The CENELEC standards [5–7] which apply to certification of safety critical railway applications qualify the use of formal methods as highly recommended. This is one of the main drivers for the use of formal methods in the railway domain.

In this paper we show how Event-B can be used to support the verification and validation in the safety critical railway domain and demonstrate this on a simple railway model. Section 2 presents the aforementioned railway model used for demonstration purposes, Sect. 3 explains some principles of formal modeling applied to the demonstration model and Sect. 4 discusses aspects of verification and validation of presented model facets as well as perceived limitations. Finally, Sect. 5 concludes the paper and outlines possible future work.

2 Railway Model

The model presented in this paper is that of a simplified interlocking system consisting of the following basic elements:

Rail Element is a unit which provides a physical running path for the trains.

Rail elements are e.g. tracks, points or crossings. A rail element consist of one or more rail segments, e.g. representing different legs of points or crossings.

Rail Connector is a port of a rail element defining the element’s connectivity.

Track is a simple rail element which connects to other rail elements on two ends (or only on one end in case of a station boundary).

Point is a movable rail element, which connects three rail elements in two different ways (tip-to-left or tip-to-right).

Crossing is an intersection of two tracks.

Signal is a device capable of passing indication whether a route may be entered by the train. A signal may be opened (green light meaning proceed) or closed (red light meaning stop).

Route is a safe running path for the train. A route consists of a directed series of connected rail elements and is protected by a signal, which allows the train to enter the route only if the route is correctly set up. Setting up the route means to bring all points into the correct position and ensure that the route does not interfere with any other route (interlocking functionality). Only after all prerequisites are fulfilled, the protecting signal may be opened and the train is allowed to enter the route.

Track Vacancy Detection determines if a particular track section is vacant (free) or occupied. A track vacancy detection section is a scope of one track vacancy detector, which may span multiple rail segments. As soon as the train enters the route, the protecting signal is closed and all track elements being first occupied and then freed again may be released to be used by another route.

The demonstrated model is reduced to the essential, yet is complex enough to showcase modeling techniques in the domain. The goal of this model is to exhibit important principles without going into the details rather than to provide a complete working application.

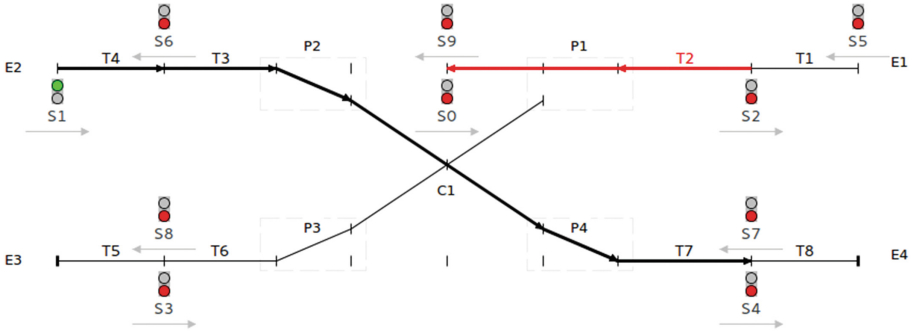


Fig. 1. Model animation of Event-B railway domain model (Color figure online)

Figure 1 shows the model using model animation on a small example. It shows a route being set (the bold line from signal S1 to signal S4 via the elements T4, T3, P2, C1, P4 and T7) and the entry signal being already opened (green light at signal S1). This route prevents the setting of any conflicting route (e.g. for the train waiting in front of signal S9 on the occupied tracks T2 and P1 shown in red).

The interlocking rules (e.g. setting and releasing the route, signaling, etc.) are the same for all stations operated by a single operator. On the other hand there is a huge variety among the railway stations – they differ in the layout and topology, but also in the specific behavior of particular elements. Therefore, it is beneficial to differentiate between the generic application describing the interlocking behavior on one side and specific applications built on top of the generic application adding the particular station data on other side.

The generic application is formulated as a railway model making assumptions about the station data (data model), but verified in a generic way, i.e. independent of any particular station data. The specific application must then assure that the provided station data meet the assumptions, i.e. are compliant to the data model. If so, then the verification of the generic application applies for all specific applications as well without having to be done again.

3 Formal Modeling

The B-Method [1] is a tool-supported formal method based on an abstract machine notation and was originally developed by Jean-Raymond Abrial. It has already been used in industrial applications such as the Paris Métro Line

14 [4, 10]. Event-B [2] is a method for system-level analysis and modeling based on B, and has been utilized in several case studies and industrial applications as well [8, 11].

Event-B is based on set theory and first order logic and is implemented in the Rodin tool chain. In addition to the Rodin core we use ProB and BMotion Studio for model checking, visualization and animation, as well as Atelier B Provers and SMT Solvers.

The demonstration railway model presented in Sect. 2 is structured in layers, each layer extending and refining the previous ones:

1. Definition of rail entities and railway topology.
 2. Distinction between the fixed and the movable rail elements and notion of rail element movement and position.
 3. Track vacancy detection.
 4. Signals.
 5. Definition of paths within the topology and the topology traversal.
 6. Signal dependency constraining the ability to set the signal depending on the status of path following this signal.
 7. Description of path life cycle, i.e. the sequence of steps necessary to construct a valid path and also dissolve is properly when not needed any more.
- ...
- * Demo Station, which supplements data for a particular station.

The layers numbered 1–7 (additional layers can be introduced for additional model functionality) define the generic application, the last refinement layer represents the specific application and is used for the visualization and animation.

The presented model defines 75 axioms and 17 invariants, which lead to 126 proof obligations, most of which are proven automatically (3 proven manually). Note that the number of proof obligations is independent of the station’s size, but is determined solely by the model complexity.

3.1 Data Model

The data model defines the assumptions about the station data as axioms and leaves the constants representing the data of the particular station abstract. The last refinement representing the specific station binds the constants to specific values conforming to the axioms. Such a formal description of the station data is stronger (compared to e.g. an XML scheme) due to the expressiveness of the first order logic and also to the behavioral specification found in the same model.

The example axiom below states that two overlapping sections are not possible:

TVD_SECT_SGMT_disjunct:

$$\begin{aligned}
 & \forall sect1, sect2 \cdot sect1 \in TVD_SECT \\
 & \wedge sect2 \in TVD_SECT \\
 & \wedge sect1 \neq sect2 \\
 & \Rightarrow TVD_SECT_SGMT(sect1) \cap TVD_SECT_SGMT(sect2) = \emptyset
 \end{aligned}$$

The constant TVD_SECT represents the track vacancy detection sections in the particular station and the constant TVD_SECT_SGMT is the assignment of rail segments to the respective track vacancy detection sections.

3.2 Functional and Safety Properties

The functional properties are described as state machines, where variables represent the current state of the railway station and events represent the state change, i.e. define the system behavior. Invariants impose restrictions of the valid state due to the domain model as well as safety constraints. Below we provide an example on how functional and safety analyses led to certain refinements of the model.

During the functional analysis a feature was identified requiring the route release functionality:

```

Event  $rem\_PATH\_CURR \hat{=}$ 
any
   $path$ 
where
  path-valid:
     $path \in PATH\_CURR$ 
then
  PATH_CURR-value:
     $PATH\_CURR := PATH\_CURR \setminus \{path\}$ 
  RAIL_ELEM_PATH_CURR-value:
     $RAIL\_ELEM\_PATH\_CURR := RAIL\_ELEM\_PATH\_CURR \triangleright \{path\}$ 
end

```

This event removes a path from the set of current paths and also releases all used railway elements occupied by this path. This event requires only that the path to be removed is set.

The variable $PATH_CURR$ contains the set of current paths and the variable $RAIL_ELEM_PATH_CURR$ identifies the path a particular rail element belongs to. For each defined path the constant $PATH_CTOR_BEG$ determines the connector at which the path begins. The symbol \triangleright denotes the range subtraction operator. It returns a subset of the left side term (being a relation), excluding mappings to elements of the right side term.

During the hazard analysis a safety property was identified requiring that a signal may be opened only if there is a correctly set up path behind this signal.

```

path-depend:
   $\forall sig \cdot sig \in SIGNAL$ 
   $\wedge SIGNAL\_ASPECT\_CURR(sig) \neq SIGNAL\_ASPECT\_DEFAULT \Rightarrow$ 
  (
     $\exists path \cdot path \in PATH\_CURR$ 
     $\wedge SIGNAL\_CTOR(sig) = PATH\_CTOR\_BEG(path)$ 
  )

```

This invariant states that for all open signals, there must be a path beginning at this signal (at the connector this signal is associated to).

The constant $SIGNAL$ represents the signals in the particular station and the constant $SIGNAL_CTOR$ the association of the signals to the respective rail connectors. The variable $SIGNAL_ASPECT_CURR$ contains the current signal aspect with the default value being $SIGNAL_ASPECT_DEFAULT$ representing default safe state being closed.

In order to fulfill the above invariant an additional constraint to the event rem_PATH_CURR becomes necessary. This is introduced by the following refinement:

Event $rem_PATH_CURR \hat{=}$

where

path-depend:

$$\begin{aligned} & \forall sig \cdot sig \in SIGNAL \\ & \wedge SIGNAL_ASPECT_CURR(sig) \neq SIGNAL_ASPECT_DEFAULT \\ & \Rightarrow SIGNAL_CTOR(sig) \neq PATH_CTOR_BEG(path) \end{aligned}$$

The new guard disallows the path removal if the signal at the path beginning is open. After adding this guard to the event the signal-related invariant is valid again.

3.3 Liveness Properties

An interlocking system is an inherently parallel system, therefore, it is meaningful to reason about its liveness properties and include a subset thereof into the formal model. There are some properties (e.g. guaranteed time bounds) which are architecture dependent and hence not modeled. Other properties can be expressed in terms of model elements and can thus be included into a formal model. The most important properties are:

Convergence stating that several events refining one abstract event converge (no oscillation occurs). This property can be expressed in Event-B by marking the events as *convergent* and by defining suitable variants. The corresponding proof obligations are then automatically generated and may be proven.

Deadlock freedom assuring that for every execution path progress is possible. This property must not only hold globally, but also for a group of events handling one external stimulus and requires that at least one event of that group is enabled. It can be expressed as a predicate stating that the disjunction of guards of all events in that group is \top (tautology). Such a proof obligation cannot be generated automatically, but it can still be expressed with Event-B means.

Predictability assuring that the execution path is predictable. This property must not only hold globally, but also for a group of events handling one external stimulus and requires that at most one event of that group is enabled. It can be expressed as a predicate stating that the pairwise conjunction of

guards of all events in that group is \perp (contradiction). Such proof obligation cannot be generated automatically, but can still be expressed with the Event-B means.

Causality adding the ability to reason about a time line. This property can be expressed using temporal logic. However, currently Event-B cannot express temporal properties.

Progress ensuring that the modeled functionality is eventually fulfilling its service obligations. This property can be formulated using suitable (weak or strong) temporal logic fairness assumption.

4 Verification and Validation

The general steps of verification and validation as we intend to use them on formal models are depicted in Fig. 2.

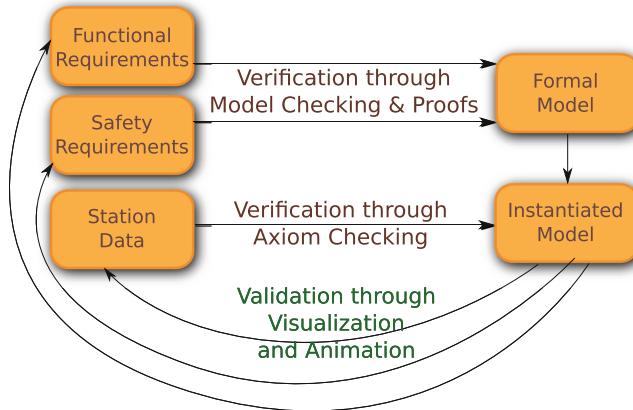


Fig. 2. Verification and validation

4.1 Verification

The verification process shall ensure that the model is unambiguous and complete. Formal models allow automatic reasoning, e.g. model finding (search for counterexamples), model checking, and model proof.

The functional and safety properties are verified semi-automatically through formal proof with Event-B. Structuring the model into independent modules reduces the scope of changes to single component only (provided the interfaces remain stable), and also the prove obligation generator invalidates only those proofs that are affected by the model modification, subsequently reducing the overall verification efforts.

In addition, the separation of the station data from the application behavior means that the verification of generic application has to be performed only once

and is applicable for all stations (provided the data meet the assumptions). Hence only the verification of the specific application (data compliance) has to be done for each station.

The verification of particular station data as a specific instance of the data model representing the respective station is achieved with the automatic compliance check of the provided data against the formulated axioms.

The verification of liveness properties in our model is still subject to research. Some of these are expressible as additional invariants and can be verified as such (by model proof with Event-B); others need the expressiveness of LTL (Linear Time Logic) properties and consequently have to be verified by model checking.

4.2 Validation

The validation process shall ensure that the model is compliant to the customer requirements. It means that the functional properties express the required behavior, the safety properties express constraints derived from the hazard analysis and the data model is neither too general nor too restrictive, i.e. all particular stations can be expressed as instances of that model.

Despite the high degree of automation a complete “push-button” verification and especially validation is not feasible. Therefore it is essential to support the domain experts so that they can perform their tasks effectively. In this process, domain specific modeling, model visualization and animation (see Fig. 1) are of major assistance.

Using a precisely formulated formal model with exactly defined semantics of all artifacts for the validation purposes aids the domain experts. Additional benefits could be gained by addressing the following limitations:

- A high level language can be built on top of Event-B allowing domain experts to formulate the problem in “their” language (DSL – Domain Specific Language), still utilizing the strong mathematical background of the Event-B method.
- Object oriented constructs can help to bridge the gap between the domain model and the “technical” model eliminating the need of mental mapping between them, thus reducing the modeling and validation effort, as well as increasing the overall confidence in the model’s correctness.
- Impact analysis of modifications allows performing delta verification and validation. Continuous verification and validation is beneficial in case of a feature driven development approach, especially when building a whole product family (product line) based on a common core.
- Traceability between the various artifacts (such as elements of a formal model) including (but not limited to) the requirements, test cases, design and implementation items is a prerequisite to the successful assessment.
- Report generators should allow incorporating proof obligations, proofs, as well as model checker results and found counterexamples into the verification and validation reports. This would save manual efforts and costs and would improve the traceability.

- For the broad utilization several other factors must be considered – interoperability and integration with other tools, modularization and namespaces, scalability, teamwork ability and industrial usability in general.

We have analyzed the iUML-B plugin for Rodin, which provides diagrammatic editors for Event-B. It allows to define data entities and their relationships as well as to model the behavior as a collection of hierarchical state machines. Some extra modeling features are also provided, like lifting of behavior to a set of instances in an object-oriented way or sequencing of events. Diagrams and the state machine animator help to visualize models and the translator generates Event-B code automatically.

Our next step is to evaluate, if the (deliberate) restriction in the iUML-B expressiveness (in comparison with the pure Event-B) poses significant limitations and how the results of subsequent steps (like proofs, model checking and animation) can be mapped back into the iUML-B notation.

4.3 Implementation and Testing

The general principles of formal model implementation and model based testing are illustrated in Fig. 3.

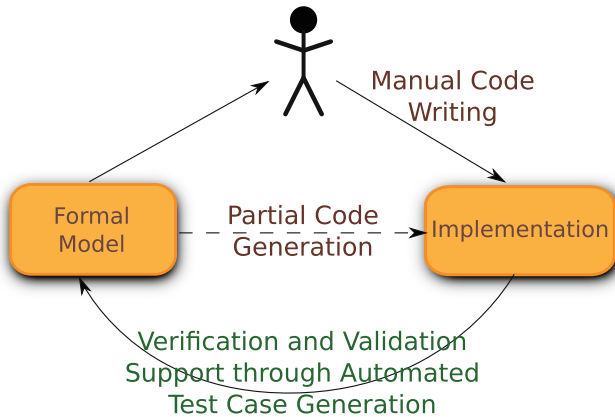


Fig. 3. Implementation and automatic test case generation

The final stage of the system development is the implementation stage. This activity involves the creation of executable components, definition of the concrete data structures, and the implementation of auxiliary functions which may have been assumed in the design and were not modeled explicitly.

Manual code writing can be supported by partial code generation, which translates the formal model into function blocks providing the implementation of the modeled aspects.

Automated test case generation helps to assure that the final implementation is compliant to the model, thus assists in both implementation verification as well as validation. There exist promising approaches to this topic, such as the MoMuT tool [3] which is currently under adaptation to support Event-B as an input language. However, care must be taken to ensure that the generated test cases deliver expressive results in order to carry the trust we gained in the model through formal methods to the implementation.

5 Conclusion

In this paper we showed how formal modeling, and in particular Event-B, can be used to assist verification and validation in the safety critical railway domain and demonstrated this with a simple railway model.

We believe that set theory together with first order logic is suitable to describe interlocking systems. The Event-B method and its implementation in the Rodin tool chain look promising, yet there is still some work to be done on the way towards an industry-ready set of tools available for the commercial usage. Moreover, there are some alternatives based on the same theoretical foundation like TLA+ (see [9]), which should also be evaluated and compared with Event-B considering not only technical, but also economical criteria.

The railway station demonstration model as presented in this publication has been released under the Eclipse Public License - v 1.0 and can be found at: <https://github.com/klar42/railground>.

Acknowledgements. The research leading to these results has received funding from the European Union’s Seventh Framework Program (FP7/2007–2013) for CRYSTAL – Critical System Engineering Acceleration Joint Undertaking under grant agreement no. 332830 and by the Austrian Research Promotion Agency (FFG) project no. 838497.

References

1. Abrial, J.R., Lee, M.K., Neilson, D., Scharbach, P., Sørensen, I.H.: The B-method. In: Prehn, S., Toetenel, H. (eds.) VDM 1991. LNCS, vol. 552, pp. 398–405. Springer, Heidelberg (1991)
2. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, New York (2010)
3. Aichernig, B., Brandl, H., Jobstl, E., Krenn, W., Schlick, R., Tiran, S.: Momut::UML model-based mutation testing for UML. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pp. 1–8. IEEE (2015)
4. 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. (eds.) FM 1999. LNCS, vol. 1708, pp. 369–387. Springer, Heidelberg (1999)
5. Cenelec European Standard: 50126-railway applications: the specification and demonstration of reliability, availability, maintainability and safety (RAMS). European Committee for Electrotechnical Standardization (1999)

6. Cenelec European Standard: 50129-railway applications: communication, signalling and processing systems - safety related electronic systems for signalling. European Committee for Electrotechnical Standardization (2003)
7. Cenelec European Standard: 50128-railway applications: software for railway control and protection systems. European Committee for Electrotechnical Standardization (2011)
8. Khuu, M.T.: Modeling a safe interlocking using the event-B theory Plug-in. Advance Project (2014)
9. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co. Inc., Boston (2002)
10. Lecomte, T., Servat, T., Pouzancre, G., et al.: Formal methods in safety-critical railway systems. In: 10th Brazilian Symposium on Formal Methods, pp. 29–31 (2007)
11. Singh, N.K.: Using Event-B for Critical Device Software Systems. Springer, London (2013)