Swansea University
Prifysgol Abertawe

Cronfa - Swansea University Open Access Repository

This is an author produced version of a paper published in:
Science of Computer Programming

Cronfa URL for this paper:
http://cronfa.swan.ac.uk/Record/cronfa37102

Paper:
Berger, U., James, P., Lawrence, A., Roggenbach, M. & Seisenberger, M. (2017). Verification of the European Rail Traffic Management System in Real-Time Maude. *Science of Computer Programming*
http://dx.doi.org/10.1016/j.scico.2017.10.011

This item is brought to you by Swansea University. Any person downloading material is agreeing to abide by the terms of the repository licence. Copies of full text items may be used or reproduced in any format or medium, without prior permission for personal research or study, educational or non-commercial purposes only. The copyright for any work remains with the original author unless otherwise specified. The full-text must not be sold in any format or medium without the formal permission of the copyright holder.

Permission for multiple reproductions should be obtained from the original author.

Authors are personally responsible for adhering to copyright and publisher restrictions when uploading content to the repository.

http://www.swansea.ac.uk/library/researchsupport/ris-support/

Verification of the European Rail Traffic Management System in Real-Time Maude

Ulrich Berger[a], Phillip James[a], Andrew Lawrence[b], Markus Roggenbach[a], Monika Seisenberger[a]

[a]*Swansea University, Swansea, UK*
[b]*Siemens Rail Automation UK, Chippenham, UK.*

Abstract

The European Rail Traffic Management System (ERTMS) is a state-of-the-art train control system designed as a standard for railways across Europe. It generalises traditional discrete interlocking systems to a world in which trains hold on-board equipment for signalling, and trains and interlockings communicate via radio block processors. The ERTMS aims at improving performance and capacity of rail traffic systems without compromising their safety.

The ERTMS system is of hybrid nature, in contrast to classical railway signalling systems which deal with discrete data only. Consequently, the switch to ERTMS poses a number of research questions to the formal methods community, most prominently: How can safety be guaranteed? In this paper we present the first formal modelling of ERTMS comprising all subsystems participating in its control cycle. We capture what safety means in physical and in logical terms, and we demonstrate that it is feasible to prove safety of ERTMS systems utilising Real-Time Maude model-checking by considering a number of bi-directional track layouts.

ERTMS is currently being installed in many countries. It will be the main train control standard for the foreseeable future. The concepts presented in this paper offer applicable methods supporting the design of dependable ERTMS systems. We demonstrate model-checking to be a viable option in the analysis of large and complex real-time systems. Furthermore, we establish Real-Time Maude as a modelling and verification tool applicable to the railway domain.

The approach given in this paper is a rigorous one. In order to avoid modelling errors, we follow a systematic approach: First, as a requirement specification, we identify the event-response structures present in the ERTMS. Then, we model these structures in Real-Time Maude in a traceable way, i.e., specification text in Real-Time Maude can be directly mapped to requirements. We explore our models by checking if they have the desired behaviour, and apply systematic model-exploration through error injection – both these steps are carried out using the formal method Real-Time Maude. Finally, we analyse ERTMS by model-checking, thus applying a formal method to the railway domain, and we mathematically prove that our analysis of ERTMS by model-checking is complete, i.e., that it guarantees safety at all times.

Keywords: Railway Signalling, ERTMS, ETCS, Hybrid Systems, Real-Time Maude, Verification, Model-Checking.

1. Introduction

In 2004, the 18th IFIP World Computer Congress identified the railway domain as a Grand Challenge of Computing Science [4] because it is of immediate concern and because it also provides a set of generic, well-understood problems whose solutions would be transferable to various other application domains, e.g., process control in manufacturing, also known as industry 4.0.

1.1. The Development of ERTMS

Reflecting on the concerns arising out of this Grand Challenge, the following are recurring statements from the 2007 Department of Transport White Paper "Delivering a Sustainable Railway" [9]:

- "Rail's biggest contribution to tackling global warming comes from increasing its capacity."
- "Reliability and capacity are amongst top passengers' concerns."
- "Increasing capacity is the most urgent investment need."

The European Rail Traffic Management System (ERTMS)/European Train Control System (ETCS) is a signalling, control and train protection system developed to address these issues. It is designed to allow for high-speed travel, to increase capacity, and to facilitate cross-border traffic movements [11]. ERTMS/ ETCS is a complex system of systems, made up of several distributed components. It is specified at four different levels, each of which defines a different use as a train control system. In our paper we consider ERTMS/ETCS Level 2, which is characterised by continuous communication between trains and radio block centres.

With the introduction of ERTMS a number of research questions arise: How can safety be guaranteed? How can reliability and performance be measured and estimated? How can capacity be measured and improved? Behind these questions are, from a computer science point of view, long-standing research challenges: how can we effectively perform qualitative and quantitative reasoning on complex systems?

- Qualitative reasoning is required to ensure safety.
- Quantitative reasoning is needed to measure both capacity and reliability.

In our paper, we provide a model of ERTMS Level 2 which we qualitatively analyse for safety using timed model-checking. It is, however, also open to quantitative analysis via simulation, for example, for studying rail network capacity (e.g. how many trains can be observed in the network within a given period of time) or exploring energy consumption (e.g. a slow train may force a faster following train to make regular speed adaptions leading to high energy consumption).

1.2. Formal Methods in the Railway Domain

Industrial standards for railway and related domains are increasingly relying on the development of formal methods for system analysis in order to establish a design's correctness and robustness. Recent examples include the 2011 version of the CENELEC standard on railway applications, the 2011 ISO 26262 automotive standard, or the 2012 Formal Methods Supplement to the DO-178C standard for airborne systems.

Fantechi [12] begins his 2014 survey on formal methods in the railway domain: "Since more than 25 years, railway signalling is the subject of successful industrial application of formal methods in the development and verification of its computerised equipment". In this context, especially the verification of *interlocking systems* has played an important role. An interlocking system is responsible for guiding trains safely through a given railway network. It is a vital part of any railway signalling system and has the highest safety integrity level (SIL4) according to the CENELEC 50128 standard.

1.3. Classical Interlocking Designs

It is still an open research question as how to perform formal safety checks on interlocking designs. The challenge is how to cope with the complexity of the problem: the state space grows exponentially in the size of the scheme plan to be verified. Several research groups, see e.g. [20, 3, 18, 14, 16, 15, 24, 23, 22, 13, 19, 39, 38, 36, 5, 28], have been addressing this challenge and have developed a number of different modelling and verification approaches.

The modelling part of such approaches usually consists of "transformations" that aim to derive a (formal) model from informal rail descriptions as used in rail industry, such as a track plan (e.g., as a CAD drawing) enriched by various tables (e.g., control tables). Similarly, the verification part usually states a safety condition (e.g., no train collision) and expresses this as a (formal) property (e.g., as a logical formula). Finally, an (automated) verification tool is utilised to provide an answer whether or not the property holds in the model. It is an open research question of how to compare such models and verification methods. As a first step in this direction, Haxthausen et al. [17] discussed the challenges, provided first steps towards a general method to compare these approaches, and performed an initial, however systematic, comparison between two of the above mentioned approaches.

Classical signalling systems based on interlockings alone treat all trains in the same way. Consequently, they are designed for worst case braking: the signalling design has to take into account the longest braking distance of a train admitted to run on the rail node to be signalled. This separates trains by long margins and reduces capacity. Concerning formal safety analysis, as such traditional systems concern discrete data only, they can be treated on a purely logical level, ignoring the aspect of time.

1.4. New Challenges Arising in ERTMS

ERTMS extends classical signalling systems by adding a radio block centre and adding control computers to trains. This makes it possible, in ERTMS/ETCS Level 2, to take into account speed and braking curves of each individual train. This data determines, for each train individually, the train's braking point well in advance of the end of the movement authority that the ERTMS signalling system has granted to the train. This allows for a separation of trains by shorter margins and thus increases capacity. In the

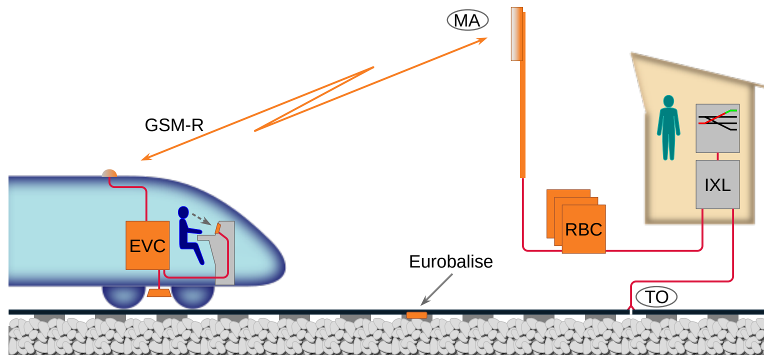


Figure 1: ERTMS/ETCS Application Level 2 core components including the on-board train computer (EVC) that detects the train's position using track-based Eurobalise sensors and communicates such information via movement authority messages (MA) with the Radio Block Centre (RBC). Decisions on allowing trains to move are then made by the Interlocking (IXL) based on this position information and readings from track occupation (TO) sensors.

ERTMS standard, the system is described using differential equations when it comes to, e.g., braking curves. Thus, in its technical definition, the system is a hybrid one. The question now is what this means for modelling: either one provides a hybrid model, or – as we do – one works with a timed model with explicit solutions of the differential equations involved. Both approaches are possible. Note that – as ERTMS Level 2 still includes interlockings – the above stated challenges on formal safety analysis for classical interlocking designs remain, however, are extended by new dimensions.

More specifically, an ERTMS/ETCS system consists of a controller, an interlocking (a specialised computer that determines if a request from the controller is "safe"), a radio block centre (the component that is responsible for the communication with the trains), track equipment, and a number of trains. While the ERTMS/ ETCS standard details the interactions between trains and track equipment (e.g., in order to obtain concise train position information) and radio block centre and trains (e.g., to hand out movement authorities), the details of how a controller, interlocking and radio block centre interact with each other are left to the suppliers of signalling solutions, such as our industrial partner Siemens Rail Automation UK. In this paper we work with the implementation as realised by Siemens Rail Automation UK. In the following we refer to this system simply as ERTMS.

1.5. Modelling ERTMS in Real-Time Maude

One development step when industry builds an ERTMS system consists of developing a so-called detailed design. Given geographical data about a specific track layout and the routes to be used, the detailed design adds a number of tables that determine the location-specific behaviour of the interlocking and radio block centre. To the best of our knowledge, our modelling of ERTMS is the first one comprising all ERTMS subsystems required for the control cycle in ERTMS Level 2 – see Figure 1 which highlights the main components of interest from the ERTMS/ETCS System Requirements Specification [2].

The objective of our modelling is to provide a formal argument that a given detailed design is safe. We focus on collision-freedom, though our model is extensible for dealing with further safety properties such as derailment and run-throughs, and possibly also with performance analysis. We also assume that all components are failure-free; that is, we do not consider fault tolerance.

We base our modelling approach on Real-Time Maude [32], which is a language and tool supporting the formal object-oriented specification and analysis of real-time and hybrid systems. In order to obtain a faithful model of ERTMS/ETCS Level 2 at the design level, we follow a methodical approach, established by the Swansea Railway Verification Group. First, we systematically identify the system's entities and determine the information flow between them – see Section 3 on the ERTMS System Architecture. Then, we give an informal description: For each of the identified components, we say which state change it has to undergo in case that an event has occurred – see Section 4 on the Event-Response Structure in this architecture. Based on this informal model, we provide a number of translations into Real-Time Maude – see Section 6 and also Figure 4 on the structure of our specifications. Static elements such as the track plan and the various tables are specified as data types using Maude equations; the ERTMS/ETCS components are represented as objects; their message exchange according to the rules laid down in the standard, or by Siemens Rail Automation UK, is captured by rewrite rules.

We note that Real-Time Maude was chosen over other modelling approaches, such as Uppaal, due to its support for differential equations and object-oriented constructs. Our modelling of ERTMS requires the explicit solving of differential equations. In addition, the object-oriented setting is natural as ERTMS is comprised of several distinct components, and thus having different objects helps to separate concerns.

1.6. Advances Compared to Our Previous Work

This paper provides a realistic model of the ERTMS extending our earlier work presented at the FTSCS workshop [26, 21]. In [26] we considered only a location-specific modelling, i.e., a fixed scheme plan was hard-coded into the model. In [21] we provided a model generic in the scheme plan; however, we still restricted ourselves to unidirectional travel and trains of zero length. In this work we overcome both these limitations. We give a generic, detailed model that fully supports bi-directional use of tracks and takes train lengths into account. Additionally, we give more details on our modelling process as described in Section 1.5.

Overall, this leads to a more complex model, especially with regards to trains. Train objects have additional fields, such as direction of travel. Also, rather than just observing that a train moves, we now capture the locations of the train's front and rear. Naturally, a more complex model poses a harder challenge when it comes to model-checking for safety.

Furthermore, we thoroughly address the question of completeness of our safety analysis, based on the criteria that Ölveczky and Meseguer give for object-oriented Real-Time Maude specifications [31]. In order to fulfil these criteria in the presence of unavoidable rounding errors, we store the maximal time elapse as an additional train component which we only recompute at discrete state transitions. This allows us to achieve time-robustness. These extra time components are only used to determine suitable sampling

time points; they do not affect the behaviour of the trains and thus do not compromise the faithfulness of our modelling of ERTMS.

Finally, we provide model-checking results based on a number of bidirectional track plans suggested as a benchmark by Haxthausen et al. [17] when comparing formal verification approaches of interlocking systems.

1.7. Organisation of the Paper

In Section 2, we introduce the ERTMS Level 2 standard, describe the detailed design of a pass-through station and briefly discuss high-level safety properties for ERTMS. We then describe our view on the ERTMS System Architecture in Section 3, and provide in Section 4 a first, informal model of ERTMS describing how the different components respond to events. In Section 5, we give a short presentation of Real-Time Maude with a focus on standard specification techniques for hybrid systems. In Section 6, we present our modelling of ERTMS in Real-Time Maude, discussing each component in detail. We formally introduce our safety properties and prove that, with finitely many time-samples, we achieve completeness in Section 7. Following this, in Section 8, we validate our model by simulation and error injection. We present our model-checking results in Section 9. Finally, in Section 10, we put our approach in the context of related work.

All Real-Time Maude models are available via the web-page of the Swansea Railway Verification Group[1].

2. ERTMS Level 2

ERTMS Level 2 extends classical railway signalling with features that aim to improve capacity and traffic management without compromising safety. To this end its location-specific design[2] extends the classical notion of a scheme plan by information used for the radio block centre. In addition to this, safety analysis for ERTMS also requires train characteristics such as maximum speed, acceleration and braking curves.

2.1. Scheme Plans

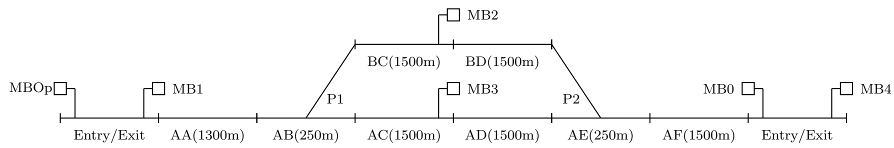
A *scheme plan* is a well-established concept within the railway domain. Figure 2 depicts such a scheme plan for a pass-through station. It consists of a track plan, a control table, release tables and RBC tables.

A *track plan* determines where and how trains can move. Trains can move along the track, in Figure 2, e.g., from the Entry/Exit on the left, over track AA to track AB. Track AB is also a point. A point can be in one of two positions: normal (travelling "straight") or reverse. Depending on the point's position, the train will either move to track BC or to track AC. Let's suppose the train moves to track BC. At the end of track BC there is a special location, indicated by a so-called marker board, in our example MB2. It is on these locations that ERTMS decides via "signalling" if a train can proceed or not. The piece of track between two consecutive marker boards (facing the same direction) is called a topological *route* (e.g., from MB1 to MB2, or from MB0 to MBOp).

In this sense, the track plan provides the topological information for the station. In our

[1] http://cs.swansea.ac.uk/rail/

[2] We focus on one ERTMS system controlling a single geographic region.



Interlocking Control Table:

Route	Clear Tracks	Normal	Reverse
1A	AA, AB, AC	P1	
1B	AA, AB, BC		P1
2	BD, AE, AF		P2
3	AD, AE, AF	P2	
4	AF, AE, AD, AC, AB, AA	P2, P1	

Interlocking Release Table:

Point	Route	Release
P1	1A	AC
P1	1B	BC
P1	4	AA
P2	2	AF
P2	3	AF
P2	4	AD

RBC Next Route Table:

Current Position	Continuation Routes (Marker Board)
Before MB1	1A (MB3), 1B (MB2)
Before MB2	2 (MB4)
Before MB3	3 (MB4)
Before MB0	4 (MBOp)

RBC Geogr. Pos. Table:

Granted Route	EoA, rel. to entry point to the left/right*
1A	3249m
1B	3249m
2	7999m
3	7999m
4	7999m*

Figure 2: Scheme Plan for a pass-through station.

example it consists of eight tracks (e.g., BC) each with a length (e.g., BC is 1500m long), six marker boards (e.g., MB1), and two points (P1, P2).

The next element of a scheme plan to consider is the so-called *control table*. The control table describes under which conditions a *route* can be set.[3] Typically, it has four columns: the first names the route under consideration; the second lists the tracks which are "free", i.e., currently not occupied by a train; the third and the fourth say which points are in normal or reverse position, respectively. For example, a train can only proceed on route 1A if point P1 is in normal (straight) position and tracks AA, AB and AC are clear, i.e., currently not occupied by a train.

The *release table* is used to implement sequential release, a technique for releasing tracks for use, in order to improve capacity. The idea is that, after a train has passed a point, this point could be used for another train already. Suppose for instance, the first train has travelled from Entry/Exit, over AA, AB to AC, where it waits for a "signal" to proceed. Then this train blocks route 1A, as track AC is occupied. However, route 1B could be made available for a second train, provided one would be allowed to move point P1 from normal to reverse. That this is actually possible is expressed by the release table. Hence, the release table describes when a point is again free to move after being locked for a particular route. For example, row one says that when sending a train on route 1A, point P1 is free to move when this train has reached track AC.

The *RBC tables* are used for similar calculations within the RBC, as discussed in Section 3. In the RBC Geographic Position Table, *EoA* (for end of authority) is the position up to which a train is allowed to travel, relative to a reference point (here to the left and right entrances of the track plan respectively).

As it is standard practice, we consider open scheme plans with entry and exit tracks. Furthermore, we make the realistic assumption that marker boards are placed at the end of tracks only. This is a typical, though not compulsory design decision taken in the railway domain. We further assume the same speed limit for all tracks. In this respect, our model is open to extensions regarding train operation, e.g., speed profiles and gradients.

2.2. Safety Conditions

In the context of ERTMS, several high-level safety conditions have been discussed, such as collision-freedom or derailment on a point. In this paper, we focus on collision-freedom, i.e., the exclusion of the possibility that two trains collide. In the context of classical signalling systems, this property is usually formulated logically, e.g., by requiring that two trains never be on the same track [22]. For ERTMS we consider, in addition, the physical invariant that the distance between trains never falls below a minimum threshold.

3. ERTMS System Architecture

Once a scheme plan has been designed, a number of control systems are implemented based around it. In the following, as a first modelling step, we systematically identify

[3] It is a design decision whether a topological route appears in the control table. The routes in the table are those available for use by trains.

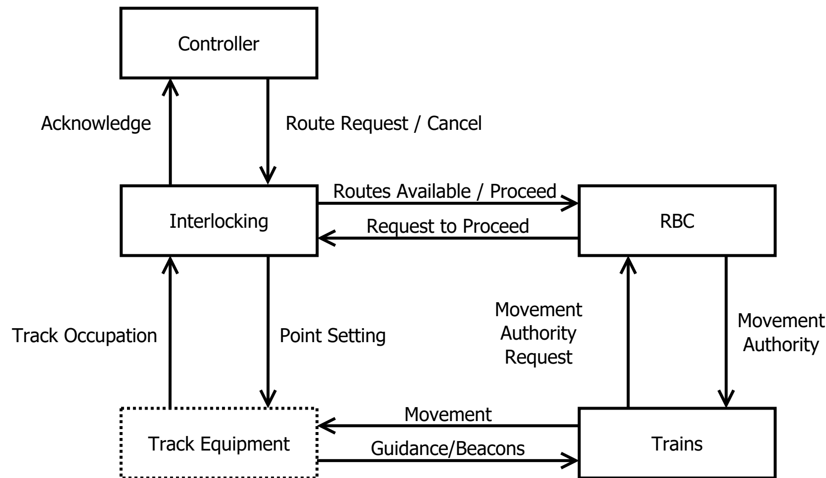


Figure 3: ERTMS control architecture.

the entities of ERTMS, describe their abstract behaviour and determine the abstract information flow between them, in line with the design by Siemens Rail Automation UK, see Figure 3.

The *controller* (manual or computerised) is responsible for controlling the flow of trains through the railway network. The controller completes this task by sending "route request" messages to the interlocking. These route requests depend on elements such as the current timetable to be adhered to and information about congestion within the network. For simplicity, we abstract from "route cancel" and "acknowledgement" messages.

The *interlocking* is responsible for setting and granting requested routes. Once the controller has requested a route, the interlocking will use information about current track occupation and point settings (from the track equipment) to determine if it is safe for the requested route to be set. Whether a route can be set or not is computed in a process based upon the conditions stipulated by the control table, see Figure 2. Once the interlocking has checked that all points on the route are free to move or already in the right position, it will send a "route available" message to the RBC. This informs the RBC that the route is free for use, however it is not yet reserved for a particular train. The RBC initiates the process of locking a route for a train by sending a "request to proceed" message to the interlocking. On receiving this message, the interlocking will then ensure that, based on the control table, all tracks for the route are free and the points are indeed locked in the required positions. Once this step is completed, the interlocking sends a "proceed" message to the RBC indicating that a train can use the route.

The RBC's main responsibility is to take the route information presented by the interlocking and use it to manage the movement of trains across geographic positions on the railway. To do this, the RBC and trains use the notion of a *movement authority*. A movement authority is an area of geographical railway that a train is permitted to move within. The furthest point along the railway to which a train is permitted to move is indicated by an EoA, which is given to a train by the RBC. The data within such

an EoA includes a distance to which the train can travel, along with a marker board towards which the train is travelling. As a train moves across the railway network, it uses beacons on the track to continually calculate its position. When it is nearing its EoA, it makes a new "movement authority request" to the RBC indicating that it would like its movement authority to be extended. After receiving this request, the RBC will map the physical location of the train to an available continuation route that has been presented to it by the interlocking.[4] This calculation is performed based on a look-up table designed as part of the RBC for a scheme plan. An example of such a table is provided in Figure 2 in the Next Route Table. It will then issue a "request to proceed" message to the interlocking for this route. Once the RBC has received a "proceed" message from the interlocking, it will compute a new EoA for the train, based on the route that has been granted. Again, this information is provided by a look-up table, see Figure 2, in particular the Geographic Position Table. This new EoA is then finally sent as a "movement authority" message to the train.

The behaviour of *trains* is parameterised by maximum speed, acceleration and braking curves. We make a maximum progress assumption for trains, i.e., trains are running as fast and as far as possible. If a train has a movement authority beyond its current position it will accelerate towards its maximum speed. When the maximum speed is reached, the train will continue to travel at this speed. Whilst accelerating or travelling at maximum speed the train will start braking at the last possible time where it is guaranteed that it will come to a halt before the EoA. Trains are guided by the track layout, respecting the positions to which the interlocking has set points. As trains move along the track, track equipment senses track occupation and reports it to the interlocking.

We assume that *track equipment* (points, track circuits, beacons etc.) functions correctly and that points move instantaneously. This is justified as our verification aim is to establish correctness of the location and train specific design parameters for an ERTMS system. Therefore, we refrain from modelling track equipment.

4. Event-Response Structure

To give a clear understanding of the types of events and corresponding messages that are involved in ERTMS, this section presents a series of tables that highlight the main responsibilities of the controller, interlocking and RBC. We refrain from presenting such a table for the train component, as trains have a much more dynamic nature than the other components which are mainly event driven. Details of this dynamic nature are presented in Section 6.

4.1. Controller Events

The main responsibility of the controller is to place route requests into the ERTMS systems. Hence it has one main event that occurs at a fixed frequency x:

Event	Conditions	Precondition
Send route request for RN to the interlocking	x seconds have passed after sending the last route request	A route RN has been randomly selected

[4] At this point, there should be maximally one route available that matches a particular train. This is ensured by the requests from the controller and also the ability of the interlocking to deny requests for conflicting routes.

4.2. Interlocking Events

The interlocking is responsible for checking and confirming the availability of routes based on requests from the controller. It is also responsible for updating the RBC with available routes, and authorising requests for trains to proceed from the RBC.

Event	Conditions	Action
Receive request from controller for route RN	RN already set	Do nothing (Request ignored)
	RN not set and (points locked or tracks not clear)	Do nothing (Request ignored)
	RN not set, points free to move and tracks clear	Set route and points, send route set message to RBC
Receive proceed request from RBC for route RN	Route is set	Lock points, unset route, send proceed granted message to RBC
	Route not set	Do nothing (Request ignored)
Receive track occupation message from train (via track equipment) concerning track TR	Front of train on TR	Update occupied tracks with track TR
	Rear of train leaving TR and TR is in the interlocking release table	Remove TR from occupied tracks and release points associated with TR
	Rear of train leaving TR and TR is not in the interlocking release table	Remove TR from occupied tracks

4.3. RBC Events

The radio block centre is responsible for the mapping between the discrete communications of the interlocking and continuous communications of the train. In particular, it responds to train requests for a new movement authority by asking the interlocking to set the route.

Event	Conditions	Action
Receive routes set message from interlocking	None	Update information stored on routes that are set in the interlocking
Receive movement authority request from a train on track TR	Follow-up route RN has been set earlier by interlocking	Send a request to proceed message to the interlocking for RN
	Follow-up route RN has not been set earlier by interlocking	Do nothing (Request ignored)
Receive proceed granted for route RN message from interlocking	None	Look-up marker board and end of authority information for route RN and send this information to the train in a 'movement authority granted' message

Each of the events and messages we have presented in this section are directly visible within our models, which are presented in Section 6.

5. Maude and Real-Time Maude

The Maude system [8] is a multi-purpose tool with support for executable specifications, simulation and verification.

Maude follows the algebraic specification approach in which sort symbols (keyword `sort`) provide a classification of the data involved and operators (keyword `op`) declare data elements (constants) and computations (functions). Hence, (part of) the data of the track-plan shown in Figure 2 can be captured as follows:

```
sort Track .
ops AA AB AC ... : -> Track .
```

The desired computation of the functions can be defined using equations (keyword `ceq` for conditional equations and keyword `eq` if there is no condition). Continuing with our example, track `AB` will be the `next` track that a train will enter after leaving track `AA`, provided the train travels towards the marker boards `MB2` or `MB3`. Variables (keyword `var`) are useful when expressing such properties and are to be read in a universally quantified manner, e.g., the next value for track `AA` is independent of the point position:

```
op next : Track MarkerBoard PointPos -> Track .

var PPos : PointPos .
var MB : MarkerBoard .

ceq next(AA, MB, PPos) = AB if MB == MB2 or MB == MB3 .
...
```

Beyond data types Maude supports the modelling of object-based systems by offering primitives to declare classes (seen as a product type consisting of a finite number of fields) and class hierarchies. To this end, Maude provides syntax for declaring classes (keyword `class`):

```
class C | a_1 : SortName_1, ... , a_n : SortName_n .
```

The above declares a class `C` with attributes `a_1` to `a_n` of sort `SortName_1` to `SortName_n`. An object of this class is represented as a term

```
< O : C | a_1 : v_1, ... , a_n : v_n > .
```

Here, object `O` has attribute values `v_1` to `v_n`.

Furthermore, there is a predefined sort `Msg` for the declaration of parametrised messages:

```
msgs M_1 ... M_k : Sort_1 ... Sort_n -> Msg .
```

Objects and messages together form the `Configuration` of an object-oriented system:

```
sorts Object Msg Configuration .
subsort Object Msg < Configuration .
```

where objects and messages are introduced as subsorts (keyword `subsort` to declare class hierarchies) of a system configuration. Objects and messages form a multi-set with `none` as the constant denoting the empty configuration and `_ _`[5] as a constructor (keyword `[ctor]`):

[5] More precisely: the space between the underscores denotes the multiset union operator, the underscores are placeholders for the operator's arguments, needed because of the infix use of the operator.

```
op none : -> Configuration [ctor] .
op _ _ : Configuration Configuration -> Configuration [ctor] .
```

A configuration itself is a subsort of a system:

```
subsort Configuration < System .
```

The data types of a system are described using (conditional) equations. In contrast, changes of configurations are written in the form of (conditional) rewriting rules (keywords `rl` and `crl`, respectively). Take for instance the following conditional rule:

```
crl  routerequest(RN1)
       < O : Inter | routeset : MAPRNB1 >
    =>
       < O : Inter | >  if (MAPRNB1[RN1] == true) .
```

This rule says that if a configuration includes a message `routerequest(RN1)` and an interlocking object `O` with attribute value `MAPRNB1`, then the message is to be deleted from the configuration provided the condition `(MAPRNB1[RN1] == true)` holds. Note that at the right hand side of the rule we do not have to repeat the attribute `routeset : MAPRNB1`, as it is not going to be changed by the rule. Finally, to explain the variable name `MAPRNB`: It is a variable for `routeset` which is a sort that maps routenames (RN) to booleans (B), thereby storing which of the routes are set.

Real-Time Maude [32, 33] extends Maude to support the specification and analysis of real-time systems. A Real-Time Maude specification consists of

- a sort `Time` (for instance the naturals, predefined as `Nat`, or the positive rationals, predefined as `PosRat`),
- a designated sort `GlobalSystem` with no subsorts or supersorts,
- a free constructor `{_} : System -> Globalsystem`, with the meaning that the term `{t}` represents the whole system,
- instantaneous (i.e., non time-consuming) rewrite rules, and
- so-called tick rules that define how time elapses (see below).

In order to express tick rules, we follow the modelling approach by Ölveczky and Thorvaldsen [34] and use their two operators `delta` and `mte`. `delta` defines the effect of time elapse on a configuration, while `mte` defines the maximal possible time elapse.

Intuitively speaking, the maximal time elapse is the maximal time that can elapse before an event has to occur. In our railway model, for instance, trains shall send a message when they have arrived on a new track. Thus, in a train model time can elapse as long as the train travels on its old track – the moment the train arrives on a new track, in a correct model, an event has to occur.

The operations `delta` and `mte` are declared as follows:

```
op delta : Configuration Time -> Configuration [frozen (1)] .
op mte : Configuration -> TimeInf [frozen (1)] .
```

Declaring a given operator, say `f`, as frozen, forbids rewriting with rules in all proper sub-terms of a term having `f` as its top operator; this avoids ill-timed rewrites. `TimeInf` is the sort `Time` enriched with an infinity element `INF`.

The two functions `delta` and `mte` distribute over objects and messages, i.e., each object has the same time available. As we will set the maximal time elapse for a message to the value `0` (see section 6.1), time can only progress once all messages have been consumed.

```
vars CON1 CON2 : NeConfiguration . var R : Time .
eq delta(none, R) = none .
eq delta(CON1 CON2, R) = delta(CON1,R) delta(CON2,R) .
eq mte(none) = INF .
eq mte(CON1 CON2) = min(mte(CON1),mte(CON2)) .
```

`NeConfiguration` is a subsort of `Configuration` denoting non-empty configurations. Both these general definitions of `delta` and `mte` will be extended to the specific objects of our Real-Time Maude specification for the ERTMS.

Finally, this allows us to describe how time elapses, whereby the maximal time elapse computed for the current configuration provides a bound for the time `R`.

```
var CURRENT : Configuration .
crl [tick] : {CURRENT} => {delta(CURRENT,R)} in time R
               if R <= mte(CURRENT) [nonexec] .
```

This rule is time-nondeterministic, as the value `R` can be chosen arbitrarily within the chosen time domain, restricted only by the given bound. Consequently, tick rules are not directly executable by the underlying Maude engine, which is marked by the `[nonexec]` attribute.

Rather than directly executing tick rules, Real-Time Maude offers the user a choice of so-called time sampling strategies:

```
(1) (set tick def 10 .)
(2) (set tick max def 10 .)
```

(1) is the *default tick mode* where time sampling is done after a fixed amount of time, here 10 seconds, at the latest, or earlier if an event occurs, i.e., if the maximal time elapse is smaller than 10. (2) uses the *maximal time sampling strategy*, i.e., it samples only when an event occurs, unless the maximal time elapse is infinite in which case the next sampling occurs after 10 seconds. The maximal time sampling strategy has the advantage that, in general, it requires fewer sampling points and thus may allow for larger systems to be verified.

In the physical model, as outlined in Section 2, time is evolving continuously according to Newton's laws. In order to control numerical errors due to rounding, we choose the sort `NNegRat` of non-negative rational numbers as our time domain. However, this domain still has infinitely many elements in the relevant bounded time interval. Thus, we discretise time by utilising the two time sampling strategies above in the hope to obtain a finite (though possibly still large) state space that allows us to apply model-checking for safety. Naturally, in this context, the question arises if safety established for finitely many time samples will imply safety for all (infinitely many) time points, i.e., if the time sampling strategies are complete. Ölveczky and Meseguer give criteria (time-robustness and tick-stabilisation) for when this is the case [31]. We will show in Section 8 that these criteria are indeed satisfied by our modelling.

For formal verification, we make use of the Maude LTL model-checker [10]. In particular, Real-Time Maude provides timed and untimed model-checking commands to check whether each behaviour from a given initial state satisfies a given temporal logic formula.

Timed model-checking is implemented using timed rewrite sequences and properties can be checked for both unbounded and bounded time periods. In particular, we will use the following command:

```
mc initState |=t phi in time < T .
```

which checks that the formula `phi` holds for all points in time up to point `T` with respect to the current tick mode. For model-checking without a time limit Real-Time Maude offers the command

```
mc initState |=t phi with no time limit .
```

which checks that the formula `phi` holds for all points in time with respect to the current tick mode. When one is interested in verification rather than counterexamples including time-stamps, one usually applies untimed model-checking rather than model-checking without time limit:

```
mc initState |=u phi .
```

Untimed model-checking essentially ignores time stamps when model-checking, but takes into account the time sampling strategy when applying the tick rules, see [10] for more details. All three approaches will return a counterexample if the formula does not hold.

6. Modelling ERTMS in Real-Time Maude

In the following, we provide an overview of our model. Figure 4 illustrates the structure of our specification. First we discuss the static data types (context) and messages used in our model; then we look at the instantaneously reacting sub-systems, i.e., controller, interlocking, and RBC; finally, we describe how we capture train behaviour, which requires differential equations describing motion. We note that our model is generic, with location-specific data as a parameter. This location-specific data has been encoded manually, however this process could be automated, for example, within the OnTrack toolset [25].

6.1. Datatypes: Location-specific Data and Messages

We model the rail topology as a connected collection of tracks, points, and routes and provide a systematic translation into Maude. For the example given in Figure 2, the location-specific data is encoded in Maude as follows:

```
sort Track .            ops AA AB AC ... : -> Track .
sort Point .            ops P1 P2 : -> Point .
sort RouteName .        ops RouteName1A ... : -> RouteName .
sort MarkerBoard .      ops MB1 MB2 ... : -> MarkerBoard .
```

The connection between tracks is given by a `next` function. For each track the next track that a train will travel along depends on the direction in which a train enters the track. A train's direction is determined by the marker board that the train is currently travelling towards. If the track in question is a point, e.g., track `AB`, it has more than one potential successor. For example, if a train is travelling from left to right, then the tracks `AC` and `BC` are both possible successors, depending on the current setting of the point.

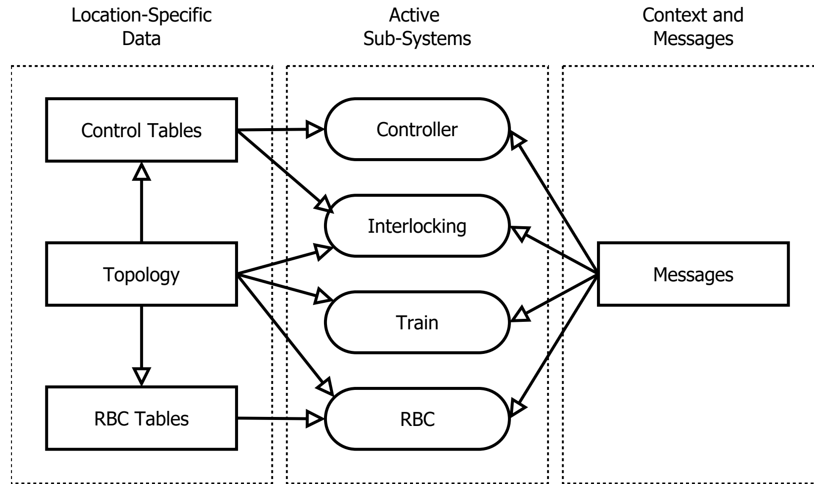


Figure 4: Structure of our specifications.

```
sort PointPos .
ops normal reverse : -> PointPos .
op next : Track MarkerBoard PointPos -> Track .

var PPos : PointPos .
var MB : MarkerBoard .

ceq next(AA, MB, PPos) = AB if MB == MB2 or MB == MB3 .
ceq next(AB, MB, normal) = AC if MB == MB3 or MB == MB4 .
...
eq next(AA, MBOp, PPos) = Exit .
```

The various tables (clear and release tables for the interlocking and the tables of the RBC) are encoded by defining a function for each column. A typical example is the “Clear Tracks” column of the control table in Figure 2:

```
op clearTracks : RouteName -> SetOfTracks .
eq clearTracks(RouteName1A) = (AA, AB, AC) .
...
eq clearTracks(RouteName4) = (AA, AB, AC, AD, AE, AF) .
```

The ERTMS components exchange a number of messages, see Figure 3. As we are dealing with a single geographic region, controller, interlocking, and RBC are unique. Thus, for most messages no object identifier is needed:

```
msgs routerequest proceedrequest ... : RouteName -> Msg .
```

This is in contrast to messages involving trains. For instance, the message

```
msg magrant : Oid MarkerBoard Nat -> Msg .
```

grants a movement authority (encoded as a natural number determining the position to which the train is allowed to travel along with marker board information) to a specific train with an object identifier of type `Oid`. Finally, as already specified in the previ-

ous section, we consider messages as urgent, i.e., they will be read immediately; their processing time/maximal time elapse is set to zero.

```
eq mte(M:Msg) = 0 .
```

6.2. Controller

An ERTMS controller issues route requests to the interlocking. Section 4.1 gives a full description of the events and messages that are involved in this process. In order to make route requests after a given time interval has passed, the controller contains a counter that is used to initiate a route request message being generated. Similarly, for efficiency reasons when it comes to model-checking, we also have a flag that indicates when the controller can stop making new requests. Therefore, a controller is modelled by an object of the following class `Controller`, whose attributes `counter` and `end` denote, respectively, the time until the next route request and a Boolean indicating whether the controller has stopped working.

```
class Controller |
 counter : NNegRat, --- Time until next route request.
 end : Bool .       --- Once true controller can stop requests.
```

The following equations demonstrate behaviour of a controller that is still in action (flag `end = false`) and has `N` seconds left before issuing the next route request. In case time progresses by an amount `R`, where `N<=R`[6], a `routerequest` message will be sent (with a randomly generated or chosen route as argument, depending on the strategy explained below) and the counter will be reset. In case `N>R` holds, the counter will be updated to `N-R`. (That is `N monus R` in Maude, which is the maximum of `N-R` and `0`.)

```
ceq delta(< O2 : Controller | counter : N, end : false >, R )
    =
    routerequest(randomRoute)
    < O2 : Controller | counter : (requestTime monus (R monus N)) >
    if (N <= R) .
ceq delta(< O2 : Controller | counter : N, end : false >, R)
    =
    < O2 : Controller | counter : (N monus R) >
    if (N > R) .
```

For a general safety analysis, a *random controller* that can request routes in any order is considered: `randomRoute` is given by

```
op randomRoute : -> RouteName .
rl randomRoute => RouteName1A .
...
rl randomRoute => RouteName4  .
```

and can non-deterministically become any possible route. Alternatively, it is possible to perform safety analysis relative to a specific strategy, for example, as given by a train timetable; for instance here is the route order of a *round-robin controller* that requests routes as follows – `1A` first, followed by `1B`, until route `4`, starting over with `1A` again:

[6] In this case actually `R=N` will hold as R is bound by the maximal time elapse of the controller which is `N`.

```
eq routeOrder = (RouteName1A : RouteName1B : ... : RouteName4) .
```

For further details we refer to the implementation. The computation of the maximal time elapse ensures that there is indeed a sampling point when the counter becomes 0 and a new route is to be requested.

```
eq mte(< O1 : Controller | counter : N, end : false >) = N .
eq mte(< O1 : Controller | counter : N, end : true >) = INF .
```

Finally, for the purpose of optimising verification, the flag `end` will be set to `true` when an 'end' message sent by a train arrives which, in the case of two trains and when checking for collision-freedom, can be sent as soon as one train has left the railway plan – i.e., no collision can happen anymore.

```
rl endmsg(O)  < O2 : Controller | end : false  >
   =>         < O2 : Controller | end : true  > .
```

6.3. Instantaneously Reacting Sub-Systems

The processing times of an interlocking and RBC are negligible compared to the time that it takes a train to pass a track. Thus, in our modelling we assume that these components react instantaneously when a message arrives, specified for the interlocking as:

```
eq mte(< O1 : Interlocking | >) = INF .
```

Interlocking. In rail control systems, the interlocking provides a safety layer between controller and track. To this end, it monitors the physical rail yard by storing the relevant information within its fields:

```
class Inter |
  occ : MapTrack2Bool,                 --- Track occupation.
  pointPositions : MapPoint2PointPos. --- Point positions.
  pointslocked : MapPoint2Bool,        --- Points locked by a route.
  routeset : MapRouteName2Bool,        --- Currently set routes.
```

`occ` stores which tracks are currently occupied (by mapping tracks to booleans), `pointPositions` stores for each point whether or not it is in normal or in reverse position, `pointslocked` stores whether a point is currently locked by a route, and finally, `routeset` stores which routes are currently set.

The interlocking is a passive component, i.e., only upon receiving a message it possibly changes its state and/or sends a message. Section 4.2 gives a full description of the events and messages that it reacts to and generates. A typical action of the interlocking is to decide if a route request from the controller is safe to grant:

```
crl  routerequest(RN1)
     < O : Inter |  routeset : MAPRNB1,
                    occ : MAPTB1,
                    pointslocked : MAPPB3 >
   =>
     < O : Inter | > if (not checkClear(RN1, MAPTB1)) or
                         pointsLocked(RN1, MAPPB3) .
```

A route request by the controller is ignored in the case that the tracks specified in the clear table for route RN1 are occupied or the points of route RN1 are locked in different positions.

If the route being requested by the controller is already set, the interlocking does nothing and ignores the route request:

```
crl  routerequest(RN1)
     < O : Inter | routeset : MAPRNB1 >
   =>
     < O : Inter | >  if (MAPRNB1[RN1] == true) .
```

Finally, in the case that it is safe to set the route requested by the controller, the interlocking will issue a `setroutes` message to the RBC informing it that the route is now available for use by a train. For a route to be set safely, the interlocking must ensure that the points required for use by the route are not locked by another route, the tracks used by the route are not occupied, and the route is not already set:

```
crl  routerequest(RN1)
     < O : Inter | routeset : MAPRNB1,
                   pointslocked : MAPPB3,
                   occ : MAPTB1,
                   pointPositions : MAPPP1 >
   =>
     < O : Inter | routeset : setRoute(RN1, MAPRNB1),
                   pointPositions : setPoints(RN1, MAPPP1) >
     setroutes(setRoute(RN1, MAPRNB1))
     if ((not pointsLocked(RN1, MAPPB3)) and
          checkClear(RN1, MAPTB1)  and
          not (MAPRNB1[RN1] == true)) .
```

We omit details of the `checkClear` operation which checks that each track listed within the control table for the requested route is unoccupied.

RBC. The RBC mediates between requests from the trains to extend their movement authorities and the successful route requests by the controller. To this end it reconciles two different views of the rail yard: trains use continuous data to represent their position (in our model the distance from their entry point of the rail yard); the interlocking uses discrete data (track occupation, set routes, point positions etc.) in its logic. Section 4.3 gives a full description of the events and messages that are reacted to and generated by the RBC. In our model, we take a rather simplified and also abstract view on the challenges involved. Namely, we abstract the mapping between continuous and discrete data into the tables presented in Figure 2. These tables present the core information needed by the RBC for computing movement authorities. In particular, we abstract away particular implementation details regarding e.g. the encoding formats used for the data.

In our model, the RBC only holds information on successful route requests (in `availableRoutes`) and for which trains (characterised by their `Oid`) it currently has an open "request to proceed" (in `designatedRoutes`):

```
class RBC |
  availableRoutes  : SetOfRouteNames,    --- Available set routes.
  designatedRoutes : MapOid2RouteName . --- Stores which routes are
                                         --- designated to trains.
```

The RBC is also a passive system component. A typical reaction is when the interlocking sends a "proceed message" for a particular route RN. Below, the RBC sends a

new "end of authority" message to the train and removes the corresponding request from its internal state.

```
rl proceedgrant(RN)
   < O2 : RBC | designatedRoutes : MAPTRN >
  =>
   magrant(getTrain(RN, MAPTRN), markerBoard(RN),endOfAuthority(RN))
   < O2 : RBC |
    designatedRoutes : removeRoute(getTrain(RN, MAPTRN),MAPTRN) >  .
```

6.4. Trains

The `Train` class is the main entity in our model. It is heavily time dependent. We designed it, at a high level, as an automaton with four movement modes (called states), namely `stop`, `acc` (acceleration), `cons` (constant maximal speed), and `brake`. There are six transitions, `stop` $\rightarrow$ `acc` $\rightarrow$ `cons` $\rightarrow$ `brake` $\rightarrow$ `stop`, and `acc` $\leftrightarrow$ `brake`. In addition, it has fields representing the current distance, speed, acceleration/deceleration rate, movement authority, name of the marker board the train is travelling towards, current track segment which the front of the train is on, current track segment of the rear of the train, maximum speed, length of the train, maximal time elapse (see below for explanation) and, for internal reasons to make model-checking feasible, a field `end` indicating whether a train has left the rail yard. `dist` denotes the position of the front of the train. We make the assumption that trains are shorter than track segments.[7]

```
class Train |
 state : TrainState,--- One of acc, cons, brake, stop.
 dist : NNegRat,    --- Current position relative to entry point.
 speed : NNegRat,   --- Current speed, between 0 and maxspeed.
 ac : NNegRat,      --- Acceleration/deceleration rate (abs value).
 ma : NNegRat,      --- Movement authority, relative to entry point.
 mb : MarkerBoard,  --- Markerboard the train is travelling towards.
 tseg : Track,      --- Current track segment of the train's front.
 tsegR : Track,     --- Current track segment of the train's rear.
 maxspeed : NNegRat,--- Maximum speed.
 length : NNegRat,  --- Length of the train.
 mtemin : TimeInf,  --- The train's maximal time elapse
                    --- (which is the minimum of four mte values).
 end : Bool .       --- Initially false, set to true if train has
                    --- left the rail yard.
```

Train movement. We assume that a train's acceleration, seen as a function over time, is piece-wise constant, and, for simplicity, use the same absolute value for the rate of both acceleration and braking. Below, we show and explain the equations defining the operation `delta` (discussed earlier) for several movement modes and mode transitions.

Recall from Section 5 that the time `R` a system can progress is bound by the overall maximal time elapse. This overall maximal time elapse of the system is defined as minimum of the maximal time elapse of all the system's components (and the tick frequency, e.g. every second)[8].

[7] We note that we are concentrating on a proof-of-concept that all system components can be modelled at once, and therefore incorporated several simplifying assumptions. The assumption about train length is not critical; removing it would essentially just lead to further case distinctions.

[8] Later in this section we show how to compute the maximal time elapse for a train.

We start our discussion of train modelling with the situation that a train in accelerating state after time `R` still accelerates, i.e., neither has reached full speed nor needs to start braking. The equation computes the new configuration of a train `O` after time `R` from its old configuration and the data held by the interlocking `O1`.

```
*** acc=>acc
ceq < O1 : Inter | pointPositions : PointSettings >
    delta(< O : Train | state : acc,
                        dist : DT, speed : S, ac : A,
                        ma : MA, tseg : AN, tsegR : ANR,
                        maxspeed : MAX, mtemin : T1  >,
          R)
      =
      < O1 : Inter | > trackseg(PointSettings,
      < O : Train |  state : acc,
                     dist : DT + S * R + (1/2) * A * R * R,
                     speed : S  +  A * R,
                     mtemin : T1 monus R >)
      if
      ANR =/= Exit and
      S + R * A < MAX and
      DT + ((S * S) / (2 * A)) + 2 * (S * R) + R * R * A
                                           <= MA monus e .
```

We recall that it is sufficient to list only those attributes that are updated; for instance, the interlocking does not change at all, therefore in the right hand side we just list `< O1 : Inter | >` with no further attributes. The interlocking provides information about the current point settings. This is necessary as we have not modelled the track equipment explicitly. (If we had, then the information whether the points are set and whether a train has traversed to new track segment would be provided by the track equipment component rather than the interlocking.) For a moment, let us further assume that, in the period `R`, the train does not traverse to a new track segment; in that case the function `trackseg`, which will take a newly updated train object and the `PointSettings` from the interlocking, will just return the newly updated train object without further modification. Thus, we can focus on how trains objects are updated: Trains move according to Newton's laws, i.e., for trains with distance `DT`, speed `S`, and acceleration rate `A`, we have after time `R`:

$$\mathtt{S}_{\text{new}}\mathtt{(S,A,R)} = \mathtt{S + A*R}$$

$$\mathtt{DT}_{\text{new}}\mathtt{(DT,S,A,R)} = \mathtt{DT + S*R} + \tfrac{1}{2}\mathtt{*A*R}^2$$

A train's maximal time elapse `mtemin` is updated relative to its previous value (see discussion at the end of the section). The conditions ensure that the train is not on the `Exit` track (1st condition), nor has reached the allowed maximal speed (2nd condition), and also does not need to start braking, due to the end of the movement authority being close (3rd condition). The correctness of the latter can be seen as follows: If a train with speed `S` is fully braking it needs the braking time `bt(S) = S/A` to come to a halt. During that time it will have travelled the braking distance :

$$\mathtt{bd(S)} = \mathtt{DT}_{\text{new}}\mathtt{(DT,S,-A,bt(S)) - DT = S}^2\mathtt{/(2*A)}$$

The condition on the maximally allowed time `R` for a train at position `DT` with speed `S` to maximally accelerate is

$$\mathtt{DT}_{\text{new}}\mathtt{(DT,S,A,R) + bd(S}_{\text{new}}\mathtt{(S,A,R)) <= MA\ monus\ e}$$

We subtract a small amount `e`, e.g., `e = 1`, to avoid that the train would start or keep accelerating in the case that the train is already close to the end of the movement authority. An easy calculation shows that the left hand is equal to

```
DT + 2*S*R + A*R^2 + S^2/(2*A)
```

The cases 'max speed reached' or 'train needs to start braking' are covered by the next equation, which includes a state change[9] from `acc` to `cons`, respectively `brake`, and a re-computation of the maximal time elapse of the train (via the function `newmte`, see below).

```
*** acc => brake, acc => cons
ceq < O1 : Inter | pointPositions : PointSettings >
  delta(< O : Train | state : acc,
                      dist : DT, speed : S, ac : A,
                      ma : MA, tsegR : ANR,  maxspeed : MAX >,
        R)
  =
  < O1 : Inter | >
  trackseg(PointSettings,
  newmte( < O : Train |
              state : if (DT + ((S * S) / (2 * A))  +
                           2 * (S * R) + R * R  * A  > MA monus e)
                      then brake
                      else cons fi,
              dist : DT + S * R + (1/2) * A * R * R,
              speed : S  +  R * A >))
  if
  ANR =/= Exit and
  (S + R * A >= MAX or
  (DT + (S * S) / (2 * A) + 2 * (S * R) + R * R * A > MA monus e)) .
```

For illustration, we also show some of the equations for a train in the other states; the remaining (omitted) equations are similar. The first one is for a train that can start moving, as its movement authority is sufficiently large. Note that we require that the end of the movement authority is at least more than `1m` ahead, otherwise a train that just came to a standstill would start moving again.

```
*** stop => acc.
ceq delta(< O : Train | state : stop,
                        dist : DT, ma : MA, tsegR : ANR >,
          R)
     =
     newmte (< O : Train |  state : acc >)
     if ANR =/= Exit  and DT <  MA monus e .
```

The next two equations are for a train in the `cons` state. Both are similar to the acc state, but simpler due to the acceleration being `0`.

```
*** cons => cons
ceq < O1 : Inter | pointPositions : PointSettings >
delta(< O : Train | state : cons,
                    dist : DT, speed : S, ac : A,
                    ma : MA, tsegR : ANR,
                    mtemin : T1 > ,
      R)
      =
```

[9] We included the state changes into the equation to ensure that state change and possible track segment updates are done at the same time.

```
        < O1 : Inter | pointPositions : PointSettings >
        trackseg(PointSettings,
        < O : Train |  state : cons,
                       dist : DT + S * R,
                       speed : S,
                       mtemin : T1 monus R >)
        if
        ANR =/= Exit and
        (DT +  ((S * S) / (2 * A))  +  S * R  < MA monus e) .
```

The following demonstrates how a movement authority request is sent when braking starts.

```
  *** cons => brake
  ceq < O1 : Inter | pointPositions : PointSettings >
  delta(< O : Train | state : cons,
                      dist : DT, speed : S, ac : A,
                      ma : MA, tseg : AN, tsegR : ANR,
                      mtemin : T1 > ,
        R)
        =
        < O1 : Inter | pointPositions : PointSettings >
        marequest(O, AN)
        trackseg(PointSettings,
        newmte(< O : Train |  state : brake,
                              dist : DT + S * R,
                              speed : S >))
        if
        ANR =/= Exit and
        DT + ((S * S) / (2 * A)) + S * R >=  MA monus e .
```

Note that we include sending the movement authority request in the equation, again to ensure that the request, track segment update, and re-computation of the maximal time elapse are happening at the same time, thereby fixing the order in which things happen. Movement authority requests will regularly be resent in the `braking` and `stop` state, the frequency depending on the maximal time elapse of the respective states.

Once a train has reached the exit, it will send an `end` message, to be consumed by the controller.

```
  eq  delta(< O : Train | tsegR : Exit, end : false > , R)
      =
      endmsg(O) < O : Train | end : true > .

  eq  delta(< O : Train | end : true > , R)
      =
      < O : Train | > .
```

Further interaction with other components. We have already discussed some of the interactions where the train sends messages, for instance, requests a new movement authority. In the following we present the rule for a train `O` with a current movement authority `MA`, receiving a new movement authority `N` (towards the marker board `MB`).

```
  crl magrant(O1, MB, N)
      < O : Train | ma : MA >
      =>
      < O : Train | ma : N, mb : MB >  if MA + 1 <= N .
```

If a train traverses to a new track segment, it is required that the interlocking is informed and that the track segment stored in the train is updated. This is done by the function

trackseg which we already mentioned in the previous paragraph in the case that nothing needs to be done, i.e., no track segment transition takes place. Below, we first show the equation for this case, where, in more detail, after a movement neither the position of the front of the train, DT, nor the rear, DT monus L, are on a new segment:

```
op trackseg :  MapPoint2PointPos Configuration -> Configuration .
ceq trackseg (PointSettings,
    < O : Train | dist : DT,
                  tseg : AN, tsegR : ANR,
                  length : L, mb : MB >)
      =
      < O : Train | >
      if
      DT <= endof(AN, dir(MB)) and
      DT monus L <= endof(ANR, dir(MB)) .
```

In the case where the front of a train has entered a new track segment, the maximal time elapse of the train will need to be recomputed, and a message indicating the transition to the new track segment is sent to the interlocking. The function computeNext (we omit the straight forward definition) computes the new track segment from the old track segment and the information about the marker board and point settings.

```
ceq trackseg (PointSettings,
    < O : Train | dist : DT,
                  tseg : AN, tsegR : ANR,
                  length : L , mb : MB >)
      =
      newmte (< O : Train |
              tseg : computeNext(AN, MB, PointSettings) >)
      tsegtransitionFront(AN, computeNext(AN, MB, PointSettings))
      if
      DT > endof(AN, dir(MB)) and
      DT monus L <= endof(ANR, dir(MB)) .
```

Finally, if the rear traverses to a new track segment, a tsegtransitionRear message will be sent to the interlocking, which will release the old track segment.

Computation of maximal time elapse. The time R that the system can progress by is bounded by the maximal time elapse which, for a train, is computed by the function newmte as the minimum of the following cases: (1) constant speed is reached, (2) the distance to MA monus e/2 becomes smaller than the braking distance,[10] (3) the front of the train reaches a new track segment, (4) the rear of the train reaches a new track segment. We show the equations for the re-computation of the maximal time elapse and the four cases for the state acc (omitting the sort information):

```
eq newmte(< O : Train | state : TS, dist : DT, speed : S, ac : A,
                        ma : MA , tseg : AN, tsegR : ANR, mb : MB,
                        maxspeed : MAX, length : L , end : false >)
=
< O : Train | mtemin : min(mteCons(TS, DT, S, A, MAX),
                        mteMA(TS,DT, S, A, MA),
                        mteTrans(TS,DT, S, A, MA, AN, MB),
                        mteTransR(TS,DT, S, A, MA, ANR, L, MB)) > .
eq newmte(< O : Train | end : true >)
   =  < O : Train | mtemin : INF > .
eq mte (< O : Train | mtemin : T1, end : false >) = T1 .
eq mte (< O : Train | end : true >) = INF .
```

[10] monus e/2 to ensure that the train comes to a halt between MA monus e and MA independent of any rounding error.

```
(1) eq mteCons(acc, DT, S, A, MAX) = ((MAX monus S) / A) .
(2) eq mteMA(acc, DT, S, A, MA) =
    solveEq(A, 2 * S, (DT + ((S * S) / (2 * A))) - (MA monus e/2)) .
(3) eq mteTrans(acc,DT, S, A, MA, AN, MB) =
    solveEq(A / 2, S, DT - (endof(AN, dir(MB)) + e/2)) .
(4) eq mteTransR(acc,DT, S, A, MA, ANR, L, MB) =
    solveEq(A / 2, S, (DT - L) - (endof(ANR, dir(MB)) + e/2)) .
```

In case (3) we need to compute the `mte` as the maximal solution of the quadratic equation `DT + S*R + A*R*R/2 = endof(AN) + e/2` which is done by the function `solveEq`. Similarly, for (2) and (4). The value `e/2` is added to ensure that the train is indeed on the new track segment.

We also show `mteMA` for the `brake` state. In that case, it is a constant, e.g., `y = 1`, after which a new movement authority request will be sent.

```
eq mteMA(brake, DT, S, A, MA) = y .
```

`mteMA` for the `cons` is similar to the `acc` state. `mteMA` for the stop case is also constant.

Rounding errors and physical uncertainties. In order to be able to prove that our time sampling is time-robust, one condition we need to check is that the maximal time elapse of a train satisfies: (5) `mte(delta(train, R)) = mte(train) - R` for all `R <= mte(train)` (see (OO1) in Sect. 7.2). This would hold if, e.g. in the case of (3), mte(train) is determined by an exact solution `R` of the equation `DT + S*R + A*R*R/2 = endof(AN)`. However, in our modelling, time is modelled by the rational numbers (enriched with `INF`). Thus, we work with a rational approximation of the root, using Maude's built-in root function[11] for floating point numbers, and convert the result back a rational number. Overall, this implies that the sampling point is not exactly the time when the train reaches the new track segment, but when the train is approximately `e/2` metres on the new track segment (i.e., `+/- e/100` metres due to rounding).[12] This small shift of the sampling point needs to be taken into account when formulating the safety condition and showing completeness, see Section 7. Furthermore, we store the maximal time elapse in the train's state, `mtemin`, to be able to fulfil (5) exactly: Whenever time evolves by a value `R < mtemin` we subtract `R`, rather than computing `mtemin` freshly and getting to a different approximation due to the inevitable rounding. Only at points where the system undergoes an instantaneous transition, see (1)-(4) above, will new mte values be computed. Concluding, we note that there is a slight discrepancy between the exact modelling, needed to be able to apply the completeness result, and the real world situation, where we have to deal with further uncertainties (see Section 11).

7. Modelling safety and addressing completeness

In this section we define a physical and a logical property guaranteeing collision-freedom of two trains and argue, using the results in [31], that model-checking is complete for these properties.

[11] Alternatively, we could implement the Newton method in Real-Time Maude, but the results are sufficiently precise.

[12] The inaccuracy in the determination of the sampling point does not concern the position of the train, therefore this can not lead to an accumulation of errors.

7.1. Defining Collision-Freedom

It is our goal to show that trains are at least a certain minimum distance apart at all times. To this end we first define a property `nocrahshDistance` that says that the *heads* of two trains are at least `minDist` metres apart at all sampling times, see Figure 5. We then model-check this property for a large enough value of `minDist` to ensure the required effective minimum distance is respected at all times, taking into account the lengths of trains and maximal acceleration values as well as the length of time sampling intervals.

We define safety for a configuration which consists of two train objects and some no further specified `REST`:

```
eq { REST  < train1 : Train |  tseg : T1 , dist : DT1, mb : MBA>
           < train2 : Train |  tseg : T2 , dist : DT2, mb : MBB > }
   |= nocrashDistance(train1, train2)
=
   ( ( not (T1 == Entry) and not (T2 == Entry) and
       not (T1 == Exit ) and not (T2 == Exit ) )
   and ( T1 == T2 or
         T1 == next(T2, mb : MBB, normal) or
         T1 == next(T2, mb : MBB, reverse) or
         T2 == next(T1, mb : MBA, normal) or
         T2 == next(T1, mb : MBA, reverse) ) )
   implies
      distance(DT1, MB1, DT2, MB2) >= minDist .
```

where

```
eq distance(DT1,MB1,DT2,MB2) = abs(rdist(DT1,MB1)-rdist(DT2,MB2)) .

eq rdist(DT,MB) = if dir(MB) == Up
                  then DT else (TLength monus DT) fi .
```

This formula reads: a configuration with two objects `train1` and `train2` of the class `train` satisfies the parameterised formula `nocrashDistance` iff the states of the two train objects under consideration are in the relation specified in the right-hand side of the equation. `T1` and `T2` are the track segments, `DT1` and `DT2` are the positions of the two trains relative to the reference point where they started, and `MBA` and `MBB` are the marker-boards that the two trains are travelling towards respectively. `rdist` computes their distance to the reference point to the left (where `TLength` is the total length of the rail yard), and `distance` computes the distance between the trains. In the formula we check that the two trains are at least `minDist` metres apart, provided they are not on an `Entry` or `Exit` track, and provided they are on the same (`T1 == T2`) or on adjacent tracks.[13] The second condition is necessary as we model positions from a single reference point on the `Entry` track. For instance, on the track plan shown in Figure 2, we can have one train on track `BC` and another train on track `AC`, both with the same distance, though by no means colliding with each other.

The traditional logical way of guaranteeing collision-freedom is to require that two trains are never on the same track:

```
  eq { REST  < train1 : Train |  tseg : T1 >
             < train2 : Train |  tseg : T2 > }
```

[13] We note that as all tracks are greater in length than our safety distance, this definition of adjacency suffices.

```
        |= nocrashTracks(train1, train2)
             = (not (T1 == Entry) and not(T2 == Entry) and
                not(T1 == Exit) and not(T2 == Exit))
               implies not (T1 == T2) .
```

This formula defines a relation `nocrashTracks(train1, train2)` which holds iff the two trains are not on the same track provided they are both 'within' the overall railway system, i.e. neither at an entry nor at an exit.

There is one caveat to this condition: `T1` and `T2` are the track segments stored within the two trains, which, for a short while ($\leq 0.501/60$ seconds, assuming a maximal speed of $60\,m/s$), may be different from the track segments the trains are actually on. This is due to the slightly late update of a train's track segment fields mentioned at the end of Section 6.4. Therefore, the formula's conclusion `not (T1 == T2)` concerning the field values within the train objects does not necessarily guarantee that the (physical positions of the) two trains are actually on different track segments at the sampling time. However, this does not compromise safety. In our model, two trains can only be on the same track segment if the trailing train's head is on the first 0.501 metres while the head of the leading train is on the last 0.501 metres of that track segment. Otherwise, the two trains would store the same track segment after the trailing train gets its fields updated, contradicting the validity of the formula not (T1 == T2). Therefore, the actual effective distance of two trains can drop no more than 1.002 metres below the safety distance predicted by the model checking, which, is the minimal track length in the plan minus maximal train length. Since the safety distance is way larger than 1 metre, this slight drop is negligible. Furthermore, it is way smaller than the accuracy of track occupation information for real interlockings, which is in the order of ± 3 metres.

7.2. Completeness

In Section 9 we model-check the safety properties defined above using the default and the maximal time sampling strategies. An important question is whether the checking is complete, that is, whether it implies that the safety properties hold at *all* times, not only at the times sampled. For the property `nocrashDistance`, which we model-checked using the default sampling strategy, the problem to consider is that the distance of the two trains might drop between two adjacent time sampling points $r_1 < r_2$ below the allowed minimum. This case can occur when a fast train that is maximally braking (acceleration $= -a$) follows a slow train that is maximally accelerating (acceleration $= a$) and the two trains reach the same speed exactly in the middle between r_1 and r_2. In that case the distance of the two trains at time $(r_1 + r_2)/2$ is by the amount of $a((r_2 - r_1)/2)^2$ lower than at the sampling times r_1 and r_2. The lemma below shows that this is the worst possible case.

We note that we only consider the relevant scenarios, namely, that the two trains `train1` and `train2` move either towards each other or in the same direction. These scenarios are summarised in Figure 5 which shows the possibility of both a head-to-head collision and a head-to-tail collision.

Lemma. To ensure a minimum distance $d_{\min}$ between `train1` and `train2` at all times it suffices to model-check the property `nocrashDistance` for the value

$$\texttt{minDist} \ = \ d_{\min} + l_{\max} + a(\Delta/2)^2$$

where $l_{\max}$ is the maximum of the lengths of the trains, a is the maximal possible absolute value of accelerations of the trains and Δ is the maximal length of the time sampling intervals.

Proof. Let $d(r)$ be the distance of the two trains at time r. Due to our assumption that the trains do not move away from each other and the fact that our trains always 'pull', i.e. have their heads at the front, we know that $d(r) \geq d_{\mathrm{h}}(r) - l_{\max}$ where $d_{\mathrm{h}}(r)$ is the distance of the trains' heads at time r. If we successfully completed the required model-checking we also know that at every time sampling point r, $d_{\mathrm{h}}(r) \geq d_{\min} + l_{\max} + a(\Delta/2)^2$, and therefore $d(r) \geq d_{\min} + a(\Delta/2)^2$. Let r_0 be a time where $d(r_0)$ is minimal. We have to show that $d(r_0) \geq d_{\min}$. If r_0 is an end-point of the entire time interval, then it is a time sampling point and hence we are done. Otherwise, d_0 is in the interior of the time interval. Since $d(r)$ is a differentiable function of r it follows that its derivative, the relative speed of the trains, is zero at r_0. Therefore, since $2a$ is an upper bound of the absolute value of the relative acceleration of the two trains, we have $d(r) \leq d(r_0) + 2a|r - r_0|^2/2 = d(r_0) + a|r - r_0|^2$ at any time r. Since the time point r_0 is at most $\Delta/2$ away from the nearest time sampling point r' it follows

$$d_{\min} + a(\Delta/2)^2 \leq d(r') \leq d(r_0) + a|r' - r_0|^2 \leq d(r_0) + a(\Delta/2)^2$$

hence $d_{\min} \leq d(r_0)$. □

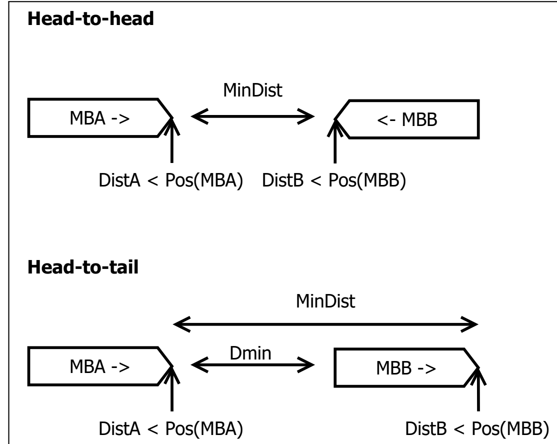


Figure 5: Illustration of safety distance between trains.

With respect to a head-to-head collision, Figure 5 highlights two trains. The first one travelling towards marker board `MBA`, with the position of the head of the train `DistA` being before the position of this marker board `Pos(MBA)`. The second train is then travelling the opposite direction towards marker board `MBB`, again with the the head of the train `DistB` being before the position of the marker board `Pos(MBB)`. In such a scenario, our modelling ensures that the heads of the trains are the full `minDist` $= d_{\min} + l_{\max} + a(\Delta/2)^2$ apart.

With respect to a head-to-tail collision, Figure 5 again highlights two trains, but this time travelling in the same direction. In this setting, we again ensure the positions of the

heads of the two trains are the full `minDist` $= d_{\min} + l_{\max} + a(\Delta/2)^2$ apart. However, notice that the safety distance between the head of the first train and tail of the second is only guaranteed to be $d_{\min}$.

Regarding the property `nocrashTracks`, one can show completeness of the maximal time elapse strategy, with respect to unbounded model-checking, by verifying the criteria given by Ölveczky and Meseguer in [31]. Essentially, one needs to prove that the property being checked is *tick-stabilising* and the system is *time-robust*. Time-robust means, roughly speaking, that the time in the system can be advanced by any amount, but an instantaneous rewrite rule can only be applied when the system has advanced time by the maximal possible amount. Tick-stabilisation means, roughly speaking, that between two sampling points the property may change its truth value at most once. This is the case for `nocrashTracks` since this property mentions only the discrete components `T1` and `T2` of the states of the trains which may change only immediately before the sampling points. Time-robustness for object-oriented systems is characterised in [31] for an object t by the following conditions on the maximal time elapse function `mte` and the function `delta` specifying the change of the configuration in the tick rule:

OO1. $\texttt{mte}(\texttt{delta}(t, r)) = \texttt{mte}(t) - r$ for $r \leq \texttt{mte}(t)$.

OO2. $\texttt{delta}(t, 0) = t$.

OO3. $\texttt{delta}(\texttt{delta}(t, r), r') = \texttt{delta}(t, r + r')$ for $r + r' \leq \texttt{mte}(t)$.

OO4. $\texttt{mte}(t) = 0$ for each left-hand side of an instantaneous rewrite rule.

OO1 and OO4 hold as can be easily seen from our modelling of ERTMS. In fact, the `mte` components of the trains were added precisely to make OO1 hold true, trivially. OO2 and OO3 can also been seen to hold, by carefully checking the definition of `delta` for objects. For example, consider the rule shown in Section 6.4 that defines `delta` for a train that accelerates and continues doing so. That OO3 holds can be verified directly, but it also follows from the fact that the speed and distance are the first and second integral of the acceleration and integration is additive. In order for the equation OO3 to hold exactly it is important that we are able to integrate exactly.

Note that the property `nocrashDistance` is not necessarily tick-stabilising since, as elaborated in the discussion of the correctness of the default time sampling strategy above, the distance of two trains may drop unnoticed below the threshold between two time sampling points unless we can control the lengths of the time sampling intervals, which we can't if we apply the maximal time elapse strategy. Hence, model-checking the property `nocrashDistance` using the maximal time elapse strategy would not provide direct evidence for its validity at all times.

8. Validation Through Simulation and Error Injection

We give a number of scenarios to illustrate that our modelling is able to capture typical errors that are made when designing ERTMS subsystems. Concerning verification tools, we rely on the model-checking capabilities of Real-Time Maude to provide the relevant counterexamples. In carrying out the verification, our starting point is that the generic models of the interlocking, RBC and trains are correct. However, we make no

assumptions about the correctness of the instantiation of our modelling with concrete design decisions, that is the *control tables*, *release tables* and *RBC tables*. Throughout the following scenarios, for ease of reading, we present all rational numbers involved in the simulations as floating point numbers rounded to two decimal places.

8.1. Simulation

We first demonstrate via simulations that trains can move as expected. Our first example concerns the behaviour of one train moving through the rail yard in Figure 2. The train first accelerates and then has to come to a standstill before the end of its movement authority. It starts on track `Entry`, with a movement authority of 3249m. The interlocking already has a preset route, thus the train can start immediately. To this end, we use the Real-Time Maude `trew` command to execute our model up to a given time bound.

```
(trew {
    < inter1 : Inter | routeset : (RouteName1B |-> true)
                       pointPositions : (P1 |-> reverse,
                                         P2 |-> reverse) , ... >
    newmte(< train1 : Train | state : acc, dist : 110, speed : 0,
             ac : 1, ma : 3249, tseg : Entry , tsegR : Entry,
             maxspeed : 60, length : 100, mtemin : 0, ... > )}
in time <= 0 .)
```

At time `0` the train is not yet moving, but the function `newmte` will compute the maximal time elapse indicating when the next state change will happen.

```
Result ClockedSystem : { < inter1 : Inter | ...>
  < train1 : Train | state : acc, dist : 110, speed : 0,
  ac : 1, ma : 3249, tseg : Entry , tsegR : Entry,
  maxspeed : 60, length : 100, mtemin : 13.37 .. >} in time 0 .
```

Thus, we look at the system again at time `14`:

```
(trew {
    < inter1 : Inter | routeset : (RouteName1B |-> true)
                       pointPositions : (P1 |-> reverse,
                                         P2 |-> reverse) , ... >
    newmte(< train1 : Train | state : acc, dist : 110, ...)> }
    in time <= 14 .)
```

That is, when the front of the train traversed to track segment `AA`.

```
Result ClockedSystem : { < inter1 : Inter | ...>
  < train1 : Train | state : acc, dist : 199.49, speed : 13.37,
    ac : 1, ma : 3249, tseg : AA, tsegR : Entry, ...
    mtemin : 6.09, .. >} in time 13.37 .
```

After a further 6.09 seconds, the rear end of the train will be on `AA`. From then on, the train accelerates until, at time `56.02`, it has to brake.

```
Result ClockedSystem :
  {marequest(train1,AB) < inter1 : Inter | ...>
  < train1 : Train | state : brake, dist : 1679.24, speed : 56.02,
    ac : 1, ma : 3249, tseg : AB, tsegR : AB, ... >} in time 56.02
```

It will issue a movement authority request 1s later.

```
{marequest(train1,AB) < inter1 : ...>
  < train1 : Train | speed : 55.02, ... >} in time 57.02
```

The system cannot progress, unless we add an RBC to our configuration.

```
(trew { < inter1 : Inter | ... > < train1 : Train | ... >
   < rbc1 : RBC | availableRoutes : empty ,
                 designatedRoutes : empty  >} in time <= 113 .)
```

As no follow-up route is available in the RBC, the train will eventually stop at 3248.49m on `BC`.

```
{< inter1 : Inter | ... > < rbc1 : RBC | ... >
 < train1 : Train | state : stop, dist : 3248.49, speed : 0,
 ma : 3249, tseg : BC, ... >} in time 112.04
```

To let the train progress, we need to add a controller to our start configuration that requests new routes:

```
< ctrl1 : Controller | counter : 10, routes : routeOrder, ... >
```

If we now let the system run, we see that the movement authority gets extended to 7999m.

```
{< inter1 : Inter | ... > < rbc1 : RBC | ... >
 < train1 : Train | ma : 7999, state : ac, mb : MB4,
   tseg : BD, ...  > < ctrl1 : ... >} in time 65
```

Finally, we add a second train to our original start configuration

```
newmte(< train2 : Train | state : stop, dist : 110, speed : 0,
ac : 1,  ma : 1, tseg : Entry , tsegR : Entry, maxspeed : 60 ,
length : 100, mtemin : 0, end : false, mb : MB0  >)
```

and find in this scenario the first train will remain on the upper route, whilst the second traverses the bottom opposite route (The first train has come to a standstill; but earlier, e.g., at time `80`, both were still accelerating).

```
{< inter1 : Inter | ... > < rbc1 : RBC | ... >
 < train1 : Train | state : stop, ma : 3249, mb : MB2,
   tseg : BC, ...  > }
 < train2 : Train | state : cons, ma : 7999, mb : MBOp,
   tseg : AF, ...  > < ctrl1 : ... >} in time 160
```

8.2. Error Injection

This section shows that our modelling is able to find errors in the design of various ERTMS components. The following scenarios use our random controller with a fixed route request interval of 25 seconds. We check the safety condition presented in Section 7.1. Furthermore, we model one slow train (max speed 20m/s) and one fast train (max speed 60m/s), as shown in the below specification. Further examples of error injections into our model, especially concerning train parameters, are presented in [26].

```
eq initState = {
  < ctrl1 : Controller |
               counter : 1,
               end : false >
  < inter1 : Inter |
              pointPositions : (P1 |-> reverse,  P2 |-> normal),
              routeset : empty,
              occ : empty,
              pointslocked : empty  >
  < rbc1 : RBC |
```

```
                availableRoutes : empty ,
                designatedRoutes : empty  >
    < train1 : Train |
                  state : stop, dist : 110,
                  speed : 0, ac : 1,
                  ma : 1, tseg : Entry,
                  tsegR : Entry, maxspeed : 20,
                  length : 100, mtemin : 1,
                  end : false, mb : MB1 >
    < train2 : Train |
                 state : stop, dist : 110,
                 speed : 0, ac : 1,
                 ma : 1, tseg : Entry,
                 tsegR : Entry , maxspeed : 60,
                 length : 100, mtemin : 1,
                end : false, mb : MB1 > } .
```

Scenario 1 – Incorrect Control Tables. We consider a scheme plan where the designer forgets to put track section AC for route 1A into the various interlocking tables in Figure 2; i.e., it is not in the clear tracks column of the interlocking control table for route 1A, and the first row of the Interlocking Release Table is missing. Applying bounded model-checking:

```
   (mc initState |=t [] nocrashDistance(train1,train2)
                                                  in time <= 300 . )
```

with the safety condition nocrashDistance highlights that two trains may be within 100 metres of each other, with both trains on track AC, highlighting a head-to-tail style violation of our safety condition.

```
   {... < train1 : Train | ac : 1, dist : 3248.5, end : false,
          length : 100, ma : 3249, maxspeed : 20, mb : MB3,
          mtemin : 10, speed : 0, state : stop,
          tsegR : AC, tseg : AC >
        < train2 : Train | ac : 1, dist : 3085.69, end : false,
          length : 100, ma : 3249, maxspeed : 60, mb : MB3,
          mtemin : 1, speed : 18.04, state : brake,
          tsegR : AC, tseg : AC > ...}
```

Scenario 2 – Incorrect RBC Tables. In this scheme plan the designer incorrectly calculates an EoA of 3449m for route 1A in the RBC tables given in Figure 2. Again model-checking with the same safety condition highlights that two trains may be within 100 metres with train1 overrunning onto track AD due to the incorrect EoA and train2 approaching on tracks AC and AD.

```
   {... < train1 : Train | ac : 1, dist : 3448.5, end : false,
          length : 100, ma : 3449, maxspeed : 20, mb : MB3,
          mtemin : 10, speed : 0, state : stop,
          tsegR : AD, tseg : AD >
        < train2 : Train | ac : 1, dist : 3287.4, end : false,
          length : 100, ma : 3449, maxspeed : 60, mb : MB3,
          mtemin : 1, speed :17.95, state : brake,
          tsegR: AC, tseg : AD > ...}
```

9. Model-Checking Results

In this section we verify a number of rail yards with the Real-Time Maude tool [33]. We start with the initial state, initState, as given in Section 8.2, which lets (only)

two trains run through the rail yards. We claim that this is justified since, for classical railway signalling, the following finitisation theorem has been established by James et al. [22]: If a signalling system is collision-free for two trains, then it is collision-free for any number of trains. We conjecture that this result carries over to ERTMS and consider our ERTMS system to be safe if – within the scheme plan under consideration – two trains will not collide. We check two invariants, both capturing the high-level condition that "trains do not collide" (c.f. Section 7.1). The first invariant captures that two trains always have to be a minimum distance apart, concretely 165m:

```
mc initState |=u [] nocrashDistance(train1,train2) .
```

or alternatively for 300 seconds[14]:

```
mc initState |=t [] nocrashDistance(train1,train2) in time <= 300 .
```

The second invariant captures the more traditional safety property that two trains are never on the same track:

```
mc initState |=u [] nocrashTracks(train1,train2) .
```

or alternatively for 300 seconds:

```
mc initState |=t [] nocrashTracks(train1,train2) in time <= 300 .
```

For these experiments, we set the frequency of controller requests to be every 25 seconds. As mentioned in Section 7.1 we use the default time sampling strategy for the first invariant and the maximal time sampling strategy for the second invariant.

As track plans, we consider those presented by Haxthausen et al. [17] as a benchmark. That is, we consider the pass-through station shown in Figure 2, which is a slight variation on the "Mini" plan presented by Haxthausen et al., as well as the "Cross" and "Twist" plans they presented, see Figures 6 and 7 respectively. These are representative in the sense that larger scheme plans can be decomposed into these smaller ones, and typical difficulties are covered.

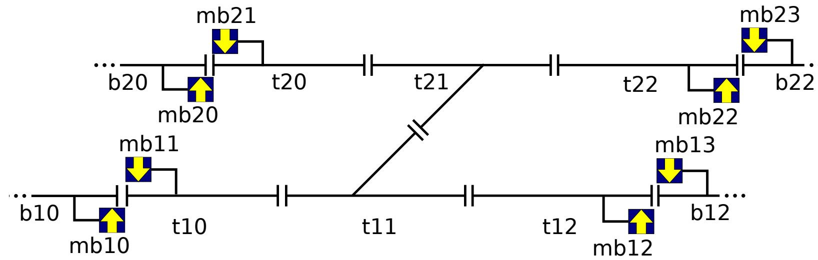


Figure 6: Track plans for the "Cross" scheme plan [17].

[14]See the discussion at the end of the section as to why this is a useful bound for bounded model-checking.

Scheme Plan	Round-Robin Controller Unbounded	
	No Crash Tracks	No Crash Distance
Pass-through	0.29s / 428508 rewrites	0.32s / 545431 rewrites
Cross	0.22s / 403997 rewrites	0.36s / 490469 rewrites
Twist	0.37s / 639841 rewrites	0.72s / 933716 rewrites

Table 1: Verification results of model-checking with restricted control strategy.

Scheme Plan	Random Controller in Time 300	
	No Crash Tracks	No Crash Distance
Pass-through	39.62s / 40,147,258 rewrites	43.30s / 58,907,793 rewrites
Cross	891.50s / 503,331,780 rewrites	632.78s / 742,640,103 rewrites
Twist	1222.79s / 652,668,124 rewrites	1038.27s / 1,023,398,631 rewrites

Table 2: Verification results of model-checking with random control strategy.

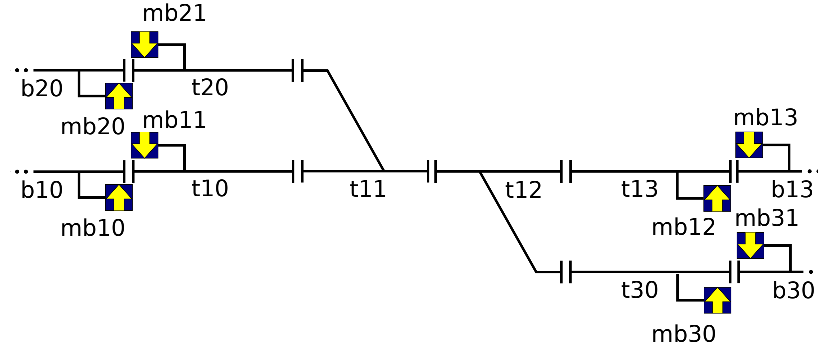


Figure 7: Track plan for the "Twist" scheme plan [17].

Verification results. In all settings the model-checking confirms that these rail yard designs are collision-free (within the given time bound, if applicable). Tables 1 and 2 show verification times[15] and the number of rewrite steps for the three rail yards against the round-robin controller and random controller (see Section 6.3).

Discussion. The results show that unbounded model-checking is successful when control is restricted, e.g., to our round-robin controller. This is due to the restrictions that such a control strategy imposes on train movements through the scheme plan. However, when using our random controller, the state space increases immensely, in particular as there are an infinite number of states possible, e.g., by the controller choosing the same route over and over again. Thus, we provide results for up to a given time bound of $300s$. This time is enough to ensure that at least one of the trains can travel completely through each of the scheme plans.

Another phenomenon is the fact that model-checking for the logical safety condition, "No Crash Tracks", requires less rewrites (approximately 20%) than for the physical safety condition, "No Crash Distance". This follows one's intuition.

[15]Using a PC running Xubuntu 14.04.2 with an i7 4790 @3.60Ghz and 32GB RAM.

As expected, model-checking times increase with the complexity of the scheme plans. One naive complexity measure would be the number of routes available in a scheme plan. We note that there are 5 routes in the "Pass-through" station, 6 routes in the "Cross", and 8 routes in the "Twist". This again follows intuition, as the random controller has more freedom in more complex track plans. This observation does not necessarily carry over to the round-robin controller where the order in which the routes are requested plays a role as well and can possibly overshadow this effect.

Finally, it is future work to consider more varied rail yards, and also how the frequency of controller requests affects model-checking results.

10. Related Work

We compare our approach to verifying ERTMS with seven other approaches, classifying them along the following three dimensions:

Comprehensiveness. This concerns the question of whether the approach discusses the overall control cycle including *all components* or if it concerns only a *subsystem* of the systems involved in the ERTMS.

Design level. This dimension asks if the artefacts to be verified are at the *requirement level* or *design level*, i.e., early in the system life-cycle, or at the *implementation level*.

Degree of formalisation. The third dimension considers if only *formal methods* are applied, or if overall the approach includes *semi-formal* elements.

Our approach covers the full control cycle between controller, interlocking, radio block centre and trains. It is at the design level. More specifically, it is our objective to verify the location-specific data of railway designs in their early development stages, accompanying a standard design process performed by signalling companies such as our industrial partner Siemens Rail Automation UK. In short: our work concerns *all components* at the *design level* and is *formal*.

Vu et al. [37] provide a generic and re-configurable model of ERTMS Level 2 interlockings at the design level. They present their model as a Kripke structure and verify high-level safety properties such as the absence of head-to-head collision or derailment on a point. They introduce the concept of virtual signals and argue that this allows them to handle the assignment of movement authorities in a similar way to the situation where conventional signals are used. The verification technology they apply is SMT solving as implemented in the RT-Tester tool-box. Overall the tool is able to scale to large Danish rail stations. Their approach abstracts from trains and the RBC and presumes these components to always behave correctly. Their verification focuses on the interlocking component at the design level. Concerning the modelling of the interlocking component, differences between Vu et al. and the work presented here appear to be mostly due to national peculiarities. Vu et al. model the Danish interlocking system, which is based upon "interlocking tables" similar to our control tables. In summary, their approach targets a *subsystem* of ERTMS at the *design level* in a *formal* way.

Cimatti et al. [7] apply software model-checking to verify the implementation level of a subsystem responsible for the allocation of logical routes to trains. The software

under consideration has been developed by Ansaldo-STS and is part of this company's implementation of ERTMS Level 2. An example of a property under consideration is "no two different trains occupy the same track". Cimatti et al. represent the software in the VELOS specification language (resembling the C++ language), the properties of interest in temporal logic, and discuss in detail the performance of different model-checkers. Their approach concerns a *subsystem* at the *implementation level* where they apply *formal* methods.

Nardone et al. [30] develop a new, rail-specific specification language, DSTM4Rail, an extension of hierarchical state machines. They employ their language DSTM4Rail to the modelling of specific functionalities of the ERTMS radio block centre. Overall the objective is to obtain a formal model of ERTMS requirements for system testing purposes. Their long term goal is integration into a model-driven development processes. This work is specialised towards quality assurance for one ERTMS component. Hence, they consider a *subsystem* at the *requirement level* only, where they apply *formal* methods.

The *openETCS initiative* [1] sets out to provide specifications that can be used for software generation for ETCS train control components, track elements, and functionality to be integrated in track side interlocking systems. This software development follows a model-driven approach, where the methods and tools aim to comply with a SIL 4 development process. This initiative offers a wide spectrum of research, which can be classified mostly as targeting a *subsystem* at the *design* or *implementation* level. In terms of methods the initiative explores a number of *semi-formal* elements.

Chiappini et al. [6] work towards the formalisation and validation of the overall ERTMS/ETCS specification. To this end, they formalise a reference subset (including movement authority management and RBC/RBC handover) of the system requirements through a set of concepts and diagrams in UML, and through additional constraints in a defined controlled natural language. This formalisation then undergoes an automatic validation check covering questions concerning consistency, scenario compatibility, and if certain properties hold. Their work puts the ERTMS/ETCS specifications themselves under scrutiny. Their work concerns a *subsystem* with the objective to be extended to *all components*; it takes place at the *requirement level* where *semi-formal* methods are used.

Platzer and Queser [35] use Differential Dynamic Logic to formalise ETCS Level 3 with free parameters (e.g. for speed and acceleration of trains). They use the verification tool KeYmaera to automatically derive and verify constraints on the parameters that guarantee correctness of the ETCS co-operation protocol regarding the interaction of the RBC with a train. Among the correctness properties considered are safety, controllability, reactivity and liveness. They also show that these properties continue to hold in the presence of perturbation by disturbances in the dynamics. This work is close to ours, however, it concerns a system of a different nature, as Level 3 utilises moving block signalling, while we are looking into Level 2 which concerns fixed block signalling. Furthermore, they deal with railways on a far more abstract level: location-specific aspects such as track plans are not considered. Hence, their work deals with *all components* and takes place on an abstract *design level* where *formal* methods are applied.

The verification of railway interlocking systems is a challenging task, and therefore several research groups have suggested to solve it by using formal methods, however using different modelling and verification approaches. To advance this research, there is a need to compare these approaches. To this end, *Haxthausen et al.* [17] suggest a way

to compare different formal approaches for verifying designs of route-based interlocking systems and demonstrate it through modelling and verification approaches developed within the research groups at DTU/Bremen and at Surrey/Swansea. In this paper, we already verify their benchmark scheme plans. However, it is left as future work to investigate the numerous scenarios that they obtain from these benchmark scheme plans by systematic error injection. This comparison concerns a specific *subsystem*, namely interlockings at the *design level* where *formal* methods are applied.

The diagram below (fig. 8) summarises and visualises the comparison.

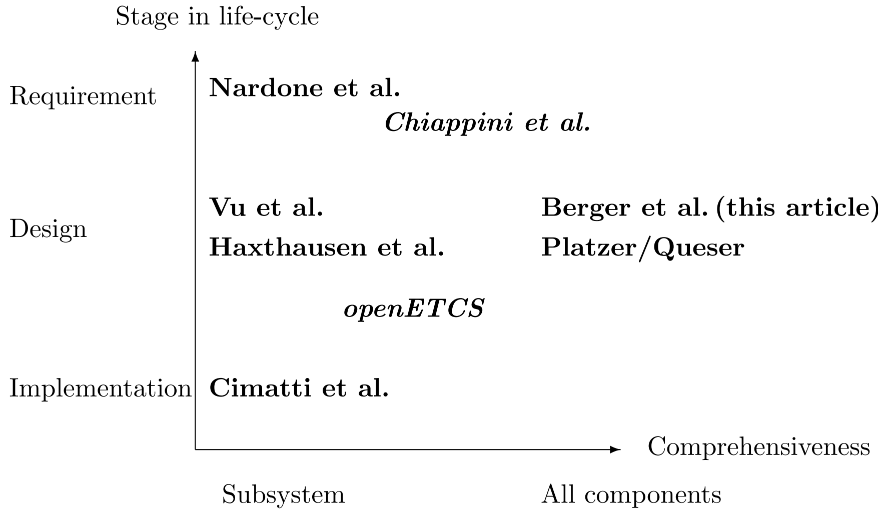


Figure 8: Comparison of **formal** and ***semi-formal*** verification approaches to ERTMS

11. Summary and Future Work

In this paper, we modelled, validated, and verified a complex system of systems of hybrid nature. We presented an analysis of the ERTMS system, described its information flow, gave an informal model by deciding on events and formulating tables stating their influence of the system state, and, finally, provided a concise formal model in Real-Time Maude. This model is astonishingly small; it consists of around 1000 lines of code. We believe that this is due to the advanced concepts, especially the object-oriented features that Real-Time Maude offers. We showed that safety analysis of our model is complete in spite of using only finitely many time sampling points. Through simulation we demonstrated that our model exhibits a number of expected behaviours. Furthermore, by systematic error injection, we showed that safety in ERTMS depends on all its components. Simulation and error injection together give us confidence that our model is valid. Finally, we presented a number of model-checking results that indicate that, for small bi-directional rail yards, model-checking of physical and logical safety properties is feasible.

In Section 8.2 on error injection, we demonstrated that the safety of ERTMS depends on the correctness of all of its components. This was achieved by showing that it is

possible to inject one mistake in the interlocking alone, or in the RBC alone, or in one train alone in such a way that safety is violated. Concerning its location-specific design, ERTMS Level 2 is not an error tolerant system: a design flaw in one of its components cannot be compensated by the other components. Though this property is to be expected, to the best of our knowledge we are the first to demonstrate it. This is thanks to the fact that our work is the first to include all ERTMS components.

Haxthausen et al. [17] compare, in the context of traditional railway signalling systems, the distinguishing power of verification approaches and present, as a benchmark, a number of simple verification problems. These include our examples "Cross" and "Switch", see Figures 6 and 7, whose verification results in an ERTMS Level 2 setting we discuss in Section 9. It would be desirable to add more comparative studies, however, the area of railway verification tends to lack such comparative benchmarks. With regards to the complexity that the verification of a railway network poses, in traditional railway signalling most authors tend to agree that the input size is best measured by giving the number of tracks, points, and signals involved. As a first approximation, verification complexity is assumed to be exponential in a weighted sum of these. However, this approximation abstracts from important topological properties such as the number of conflicting and/or opposing routes, which can have dramatic effects on verification complexity, see [29] for some examples. These considerations carry over to our ERTMS setting, as ERTMS includes traditional interlockings. However, to the best of our knowledge, there is no ERTMS specific metric available to calculate the verification complexity posed by a railway network.

In our modelling of the ERTMS and its analysis we had to deal with different kinds of inexactness at several places: As mentioned at the end of Section 6, there is a discrepancy between our exact modelling and the real world where there always remains some uncertainty about the exact location of a train. We dealt with this issue by allowing for extra space between trains. In the completeness proof in Section 7.2 we solved the problem of rounding errors in the calculation of the maximal time elapse by adding this data as an extra component to the trains. In this way we could guarantee the equations OO1-OO4 to hold exactly, as required in [31] for completeness. These phenomena of inexactness are not a peculiarity of our modelling, but something that occurs in any modelling and verification task involving continuous data. Therefore, a systematic way of dealing with these issues would be highly desirable. For example, one could try to prove approximative versions of the completeness results in [31] that tolerate controlled rounding errors.

It is future work to explore further, and in particular more complex, rail yards. On the practical side, we intend to extend our modelling with further controller strategies, especially 'taming' the random controller, and more complex train progression behaviour. On the theoretical side, we plan to prove our conjecture that for a complete safety analysis it is enough to consider two trains only – as is the case in our CSP||B models [22]. Naturally, it would be desirable to prove such a property for railways in general. However, the railway domain is an informal one and therefore not open to formal proofs. Consequently, such a result can be only proven after a formal model has been provided.

Additionally, we want to investigate abstraction techniques to reduce model-checking time: For traditional railway signalling several authors, e.g., [22, 27], have investigated compositional verification methods – such results may allow us to scale-up our method to real ERTMS networks that can hold hundreds of routes.

From an industrial perspective, Siemens Rail Automation UK considers our work to have high potential to improve quality assurance within their software development process of ERTMS Level 2 interlockings and RBCs. More concretely, they started a new research programme with us concerning model-based testing in Real-Time Maude. Objectives of this programme include: Given a location-specific design for an ERTMS controlled rail yard, i.e., interlocking tables and rules regarding messages to be sent by the RBC, provide a formally verified, location-specific test model and derive test suites from it, that can be used to test the location-specific Siemens Rail Automation UK RBC realisation and interlocking computer – either in isolation or in combination.

Acknowledgement. The authors would like to thank Simon Chadwick, Siemens Rail Automation UK, for his continued support and many helpful discussions. We also appreciate the useful feedback and advice from Peter Ölveczky and José Meseguer on Real-Time Maude. As usual, Erwin R. Catesbeiana (Jr) was keeping us on track. We are also grateful to the three anonymous referees for their helpful comments and constructive criticism. Finally, the financial support of Siemens Rail Automation UK and EPSRC (EP/P5057631) is gratefully acknowledged.

References

[1] openETCS. `http://openetcs.org`, 2017. Accessed: 2017-01-30.

[2] Alcatel, Alstom, Ansaldo Signal, Bombardier, Invensys Rail and Siemens. System Requirements Specification, Chapter 2, Basic System Description, 2006. SUBSET-026-2.

[3] M. Banci, A. Fantechi, and S. Gnesi. Some Experiences on Formal Specification of Railway Interlocking Systems Using Statecharts. In *Software Engineering and Formal Methods - TRain Workshop at SEFM 2005*, 2005.

[4] D. Bjørner. TRain: The Railway Domain - A Grand Challenge. In R. Jacquart, editor, *Building the Information Society*, volume 156 of *IFIP International Federation for Information Processing*, pages 607–611. Springer, 2004.

[5] Y. Cao, T. Xu, T. Tang, H. Wang, and L. Zhao. Automatic Generation and Verification of Interlocking Tables Based on Domain Specific Language for Computer Based Interlocking Systems. In *Proceedings of the IEEE International Conference on Computer Science and Automation Engineering, CSAE 2011*, volume 2, pages 511 – 515. IEEE, 2011.

[6] A. Chiappini, A. Cimatti, L. Macchi, O. Rebollo, M. Roveri, A. Susi, S. Tonetta, and B. Vittorini. Formalization and Validation of a subset of the European Train Control System. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, Volume 2, ICSE'10*, pages 109–118. ACM press, 2010.

[7] A. Cimatti, R. Corvino, A. Lazzaro, I. Narasamdya, T. Rizzo, M. Roveri, A. Sanseviero, and A. Tchaltsev. Formal verification and validation of ERTMS industrial railway train spacing system. In *Computer Aided Verification, CAV'12*, volume 7358 of *Lecture Notes in Computer Science*, pages 378–393. Springer, 2012.

[8] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.

[9] Department for Transport. Delivering a sustainable railway: white paper CM 7176, 2007.

[10] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. In *Rewriting Logic and Its Applications, WRLA'02*, volume 71 of *ENTCS*, pages 162–187. Elsevier, 2004.

[11] European Railway Industry. ERTMS. `http://www.era.europa.eu/Core-Activities/ERTMS/Pages/home.aspx`, 2015. Accessed: 2015-08-30.

[12] A. Fantechi. Twenty-Five Years of Formal Methods and Railways: What Next? In *Software Engineering and Formal Methods*, volume 8368 of *Lecture Notes in Computer Science*, pages 167–183. Springer, 2014.

[13] A. Ferrari, G. Magnani, D. Grasso, and A. Fantechi. Model Checking Interlocking Control Tables. In *FORMS/FORMAT 2010*, pages 98–107. Springer, 2011.

[14] A. E. Haxthausen. Towards a Framework for Modelling and Verification of Relay Interlocking Systems. In *16th Monterey Workshop: Modelling, Development and Verification of Adaptive Systems: the Grand Challenge for Robust Software*, volume 6662 of *Lecture Notes in Computer Science*, pages 176–192. Springer, 2011.

[15] A. E. Haxthausen. Automated Generation of Formal Safety Conditions from Railway Interlocking Tables. *International Journal on Software Tools for Technology Transfer (STTT), Special Issue on Formal Methods for Railway Control Systems*, 16(6):713–726, 2014.

[16] A. E. Haxthausen, M. L. Bliguet, and A. A. Kjær. Modelling and Verification of Relay Interlocking Systems. In *Foundations of Computer Software, Future Trends and Techniques for Development, 15th Monterey Workshop*, volume 6028 of *Lecture Notes in Computer Science*, pages 141–153. Springer, 2010.

[17] A. E. Haxthausen, H. N. Nguyen, and M. Roggenbach. Comparing Formal Verification Approaches of Interlocking Systems. In *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - First International Conference, RSSRail 2016*, volume 9707 of *Lecture Notes in Computer Science*, pages 160–177. Springer, 2016.

[18] A. E. Haxthausen, J. Peleska, and S. Kinder. A Formal Approach for the Construction and Verification of Railway Control Systems. *Formal Aspects of Computing*, 23(2):191–219, 2011.

[19] A. E. Haxthausen, J. Peleska, and R. Pinger. Applied Bounded Model Checking for Interlocking System Designs. In *Software Engineering and Formal Methods*, volume 8368 of *Lecture Notes in Computer Science*, pages 205–220. Springer, 2014.

[20] A. Iliasov, I. Lopatkin, and A. Romanovsky. Practical Formal Methods in Railways - The SafeCap Approach. In *Reliable Software Technologies, Ada-Europe 2014, Proceedings*, volume 8454 of *Lecture Notes in Computer Science*, pages 177–192. Springer, 2014.

[21] P. James, A. Lawrence, M. Roggenbach, and M. Seisenberger. Towards Safety Analysis of ERTMS/ETCS Level 2 in Real-Time Maude. In *Formal Techniques for Safety-Critical Systems, FTSCS 2015*, volume 596 of *Communications in Computer and Information Science*, pages 103–120, 2015.

[22] P. James, F. Moller, N. H. Nga, M. Roggenbach, S. A. Schneider, and H. Treharne. Techniques for modelling and verifying railway interlockings. *STTT*, 16(6):685–711, 2014.

[23] P. James, F. Moller, H. N. Nguyen, M. Roggenbach, S. A. Schneider, and H. Treharne. On modelling and verifying railway interlockings: Tracking train lengths. *Sci. Comput. Program.*, 96:315–336, 2014.

[24] P. James and M. Roggenbach. Encapsulating formal methods within domain specific languages: A solution for verifying railway scheme plans. *Mathematics in Computer Science*, 8(1):11–38, 2014.

[25] P. James, M. Trumble, H. Treharne, M. Roggenbach, and S. Schneider. Ontrack: An open tooling environment for railway verification. In *NASA Formal Methods, 5th International Symposium, NFM 2013, Proceedings*, pages 435–440. Springer, 2013.

[26] A. Lawrence, U. Berger, P. James, M. Roggenbach, and M. Seisenberger. Modelling and analysing the European Rail Traffic Management System in Real-Time Maude. In *FTSCS'14 – Preliminary Proceedings*, 2014.

[27] H. D. Macedo, A. Fantechi, and A. E. Haxthausen. Compositional Verification of Multi-station Interlocking Systems. In *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications, ISoLA 2016*, volume 9953 of *Lecture Notes in Computer Science*, pages 279–293, 2016.

[28] A. Mirabadi and M. B. Yazdi. Automatic Generation and Verification of Railway Interlocking Control Tables Using FSM and NuSMV. *Transportation Problems*, 4:103–110, 2009.

[29] F. Moller and M. Roggenbach. Towards Tool Interaction and Safety Assessment, RSSB, 2016. DITTO Project Deliverable 1.3 Milestone 8, available at `www.dittorailway.uk`.

[30] R. Nardone, U. Gentile, A. Peron, M. Benerecetti, V. Vittorini, S. Marrone, R. De Guglielmo, N. Mazzocca, and L. Velardi. Dynamic State Machines for Formalizing Railway Control System Specifications. In *Formal Techniques for Safety Critical Systems, FTSCS'14*, Communications in Computer and Information Science 476, pages 93–109. Springer, 2015.

[31] P. C. Ölveczky and J. Meseguer. Abstraction and Completeness for Real-Time Maude. In *Rewriting Logic and Its Applications, WRLA'06*, volume 176 of *ENTCS*, pages 5–27, 2007.

[32] P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symbolic Computation*, 20(1-2):161–196, 2007.

[33] P. C. Ölveczky and J. Meseguer. The Real-Time Maude tool. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2008*, volume 4963 of *Lecture Notes in Computer Science*, pages 332–336. Springer, 2008.

[34] P. C. Ölveczky and S. Thorvaldsen. Formal modeling, performance estimation, and model checking

of wireless sensor network algorithms in Real-Time Maude. *TCS*, 410(2):254–280, 2009.

[35] A. Platzer and J.-A. Quesel. European Train Control System: A Case Study in Formal Verification. In K. Breitman and A. Cavalcanti, editors, *Formal Methods and Software Engineering. ICFEM 2009*, volume 5885 of *Lecture Notes in Computer Science*, pages 246–265. Springer, 2009.

[36] D. Tombs, N. Robinson, and G. Nikandros. Signalling Control Table Generation and Verification. In *Proceedings of Cost Efficient Railways through Engineering, CORE 2002*, pages 415–425. Railway Technical Society of Australasia, 2002.

[37] L. H. Vu, A. E. Haxthausen, and J. Peleska. Formal Modeling and Verification of Interlocking Systems Featuring Sequential Release. In *Formal Techniques for Safety Critical Systems, FTSCS'14*, Communications in Computer and Information Science 476, pages 223–238. Springer, 2015.

[38] K. Winter. Optimising Ordering Strategies for Symbolic Model Checking of Railway Interlockings. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*, volume 7610 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2012.

[39] K. Winter, W. Johnston, P. Robinson, P. Strooper, and L. van den Berg. Tool Support for Checking Railway Interlocking Designs. In *10th Australian workshop on Safety Critical Systems and Software, SCS'05, Proceedings*, volume 55, pages 101–107. Australian Computer Society, Inc., 2006.