Towards Safety Analysis of ERTMS/ETCS Level 2 in Real-Time Maude

Phillip James[1], Andrew Lawrence[2], Markus Roggenbach[1], and Monika Seisenberger[1](✉)

[1] Swansea University, Swansea, UK
m.seisenberger@swansea.ac.uk
[2] Hitachi Data Systems, Poole, UK

Abstract. ERTMS/ETCS is a European signalling, control and train protection system. In this paper, we model and analyse this complex system of systems, including its hybrid elements, on the design level in Real-Time Maude. Our modelling allows us to formulate safety properties in physical rather than in logical terms. We systematically validate our model by simulation and error injection. Using the Real-Time Maude model-checker, we effectively verify a number of small rail systems.

1 Introduction

The European Rail Traffic Management System (ERTMS)/European Train Control System (ETCS) is a European signalling, control and train protection system designed to allow for high speed travel, to increase capacity, and to facilitate cross-border traffic movements [7]. ERTMS/ETCS is a complex system of systems, made up by distributed components. It is specified at four different levels, where each level defines a different use as a train control system. In our paper we consider ERTMS/ETCS Level 2, which is characterised by continuous communications between trains and a radio block centre.

The switch from classical railway signalling systems to ERTMS/ETCS train control poses a number of research questions for the formal methods community. Can safety be guaranteed? Can formal methods be used to confirm that such a switch improves capacity? Is it possible to predict capacity using formal methods? To address such questions it is necessary to develop and analyse timed or hybrid models. ERTMS/ETCS Level 2 takes speed and braking curves of each individual train into account. These determine the train's braking point well in advance of the end of authority that the signalling system had granted to this train. Such an approach is in contrast to classical signalling systems, which treat all trains in the same way. Therefore, they need to be designed for worst case braking. Consequently, in formal safety analysis, such traditional systems can be treated on a purely logical level, ignoring the aspect of time – see, e.g., [9,10].

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, track equipment, and a number of trains. While the ERTMS/ETCS standard details the interactions between trains and track equipment

C. Artho and P.C. Ölveczky (Eds.): FTSCS 2015, CCIS 596, pp. 103–120, 2016.
DOI: 10.1007/978-3-319-29510-7_6

(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 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. In the following we refer to this system simply as ERTMS.

One development step when building an ERTMS system consists of developing a so-called detailed design. Given geographical data such as a specific track layout and what routes through this track layout shall be used, the detailed design adds a number of tables that determine the location specific behaviour of interlocking and radio block centre. The objective of our modelling is to provide a formal argument that a given detailed design is safe. Here we focus on collision freedom, though our model is extensible for dealing with further safety properties, and possibly also with performance analysis.

We base our modelling approach on Real-Time Maude, 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 on the design level, we follow a systematic approach, established by the Swansea Railway Verification Group.

This paper extends our location-specific modelling presented in past work [12] to a generic and far more detailed modelling. It is organised as follows. First, we introduce the ERTMS Level 2 standard, and briefly discuss high level safety properties for ERTMS. Then, we give a short presentation of Real-Time Maude with a focus on standard specification techniques for hybrid systems. In Sect. 4, we present our modelling of ERTMS in Real-Time Maude, discussing each component in detail. In Sect. 5, we validate our model by simulation and error injection. Finally, we present model checking results and put our approach in the context of related work.

2 ERTMS Level 2

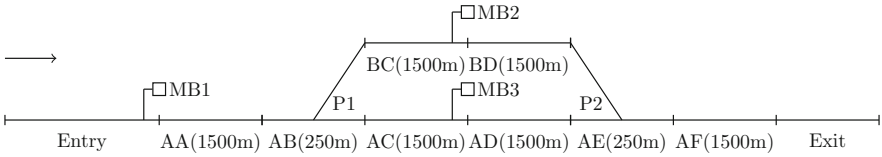
ERTMS Level 2 extends classical railway signalling. To this end its location specific design[1] extends the classical notion of a scheme plan by information used for the radio block centre (RBC). ERTMS safety analysis also requires train characteristics such as maximum speed, acceleration and braking curves.

2.1 Scheme Plans

A scheme plan is a well-established concept within the railway domain. Figure 1 depicts such a *scheme plan* for a pass-through station. It comprises of a track plan, a control table, release tables and RBC tables. The *track plan* provides the topological information for the station. It consists of 8 tracks (e.g., BC) each with a length, 3 marker boards (e.g., MB1), and two points (e.g., P2). A topological

[1] We focus here on one ERTMS/ETCS system controlling a single, geographic region.

route is a piece of railway on which a train can travel, (typically) between two marker boards (e.g., from MB1 to MB2). The *control table* describes under which conditions a *route* can be set.[2] For example, a train can only proceed on route 1 A when point P1 is in normal (straight) position and tracks AA, AB and AC are clear, i.e., currently not occupied by any train. The *release table* is used to implement sequential release, a technique to improve capacity. The release table describes when a point is again free to move after being locked for a particular route. For example, when sending a train on route 1A, point P1 is free to move already, when this train has reached track AC. This allows to send another train on route 1B before the first train has reached track AD and thus completely left route 1A. Finally, the *RBC tables* are used for calculations within the RBC.



Interlocking Tables:

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	

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

RBC Tables:

Current Position	Continuation Routes
Before MB1	1A, 1B
Before MB2	2
Before MB3	3

Granted Route	EOA
1A	3249m
1B	3249m
2	6499m
3	6499m

Fig. 1. Scheme plan for a pass-through station.

We consider open scheme plans with entry and exit tracks only. Furthermore, we assume that marker boards are placed at the end of tracks, and that the speed limit is the same for all tracks.

2.2 ERTMS System Architecture

Once a scheme plan has been designed, a number of control systems are implemented based around it. In the following we identify the entities of ERTMS, describe their abstract behaviour and determine the abstract information flow between them in line with the design by Siemens Rail UK, see Fig. 2.

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 are dependent upon elements such as the current timetable to be adhered to

[2] 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.

and details on 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 on 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 Fig. 1. 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 (Radio Block Centre). This informs the RBC that the route is free for use, however it is not yet reserved for a train. The RBC initiates the process of locking a route for a particular 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 that 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 a point known as the *end of authority* (EoA) which is given to a train by the RBC. 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.[3] 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 Fig. 1. 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, based on the route that has been granted, a new EoA for the train. Again, this information is provided by a look-up table, see Fig. 1. This new EoA is then finally sent as a "movement authority" message to the train.

With regards to *trains*, their behaviour 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. Namely, 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

[3] 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.

travel at this speed. Whilst accelerating or travelling at maximum speed the train will start braking at the last possible time in order not to overrun its 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 a ERTMS system for a single geographic region. Therefore, we refrain from modelling track equipment.

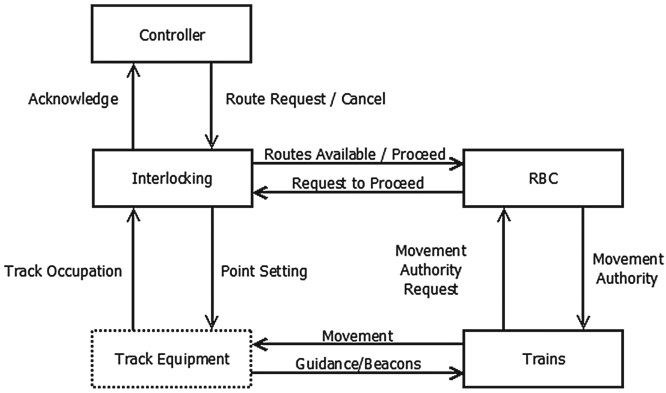


Fig. 2. ERTMS control architecture.

2.3 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., excluding the possibility that two trains collide. In the context of classical signalling systems, this property usually is formulated logically, e.g., we verify that there are never two trains on the same track [9]. In contrast, for ERTMS we rather consider the physical invariant: the distance between trains never falls below a minimum threshold.

3 Maude/Real-Time Maude

The Maude system [5] is a multi-purpose tool with support for executable specification, simulation and verification. Its wide range of capabilities made us to favour Maude. Particularly, we are interested in the Maude LTL Model Checker [6]. Real Time Maude [13] is an extension of Maude containing specific support enabling the modelling and verification of real-time systems.

Object-based systems can be modelled as multisets of objects and messages where the messages define the communication between the objects and typically trigger actions of the objects. A class C with attributes of `a_1` to `a_n` of sort `Sort_1` to `Sort_n`, and an object `O` with attribute values `v_1` to `v_n` of class `C` are written as, respectively

```
class C | a_1 : Sort_1, ... , a_n : Sort_n .
< O : C | a_1 : v_1, ... , a_n : v_n > .
```

Objects declared together with messages

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

form a multiset of the sort `Configuration`, a subsort of Maude's built-in sort `System`, using `__` for multiset union.

```
sorts Object Msg Configuration .
subsort Object Msg < Configuration .
op __ : Configuration Configuration -> Configuration [ctor] .
```

A real-time specification [13] consists of a sort `Time` (in our case `PosRat`), the constructor $\{\_\} : \texttt{System} -> \texttt{Globalsystem}$ with the meaning that `{t}` represents the whole system (and does not appear as an argument to another function - as is marked by using the independent type `Globalsystem`), instantaneous rewrite rules, and a so-called tick rule that defines how time elapses. As [17], we use the operators `delta` and `mte` in order to define the effect of time elapse on a configuration, and of the maximal possible time elapse, resp.

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

Here, `TimeInf` is the sort `Time` enriched with an infinity element `Inf`. These two functions are distributed over objects and messages, i.e., each object has the same time available, and as the maximal time elapse for a message has value 0, time can only progress once all messages are 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)) .
```

The argument `R` of type `Time` is determined by the tick rule

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

The default tick time is defined by

```
(set tick def 1 .)
```

This means we look at the configuration either at each time step, or more often in the case that some event occurs, for a justification see e.g. [15].

4 Modelling ERTMS in Maude

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/ETCS Application Level 2, c.f. Fig. 6 in the ERTMS/ETCS System Requirements Specification [2]. For simplicity, we consider only uni-directional rail yards, as these exhibit many of the components of bi-directional rail yards, but are of a lower complexity with regards to the number of routes required within the model. Also, we make the standard assumption that trains have no length. This is the typical abstraction when one deals with trains whose length is shorter than any track length in the given scheme plan. For a detailed discussion of the topic see, e.g., our publication discussing train length [10].

In the following, we provide an overview of our model:[4] first we discuss the static data types; then we look at the instantaneously reacting sub-systems, i.e., controller, interlocking, and RBC; next, we describe how we capture train behaviour, which requires differential equations describing motion; finally, we address how to express collision-freedom. We note that our model is generic, with only location specific data as a parameter. This location specific data has been encoded manually, however this process could be automated within OnTrack [11].

4.1 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 Fig. 1, the location specific data Maude is encoded as follows:

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

The connection between tracks is given by a `next` function. If the track under discussion is a point, as, e.g., track `AB`, it has two potential successors, namely `AC` and `BC`, depending on the current setting of the point.

```
op next : Track PointPos -> Track . var PPos : PointPos .
eq next(AA,PPos) = AB                                    .
eq next(AB,normal) = AC . eq next(AB,reverse) = BC       .
```

The various tables (clear and release tables for the scheme plan, the tables of the RBC) are encoded by defining a function for each column. A typical example is the "Clear Tracks" column[5] of the control table in Fig. 1:

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

[4] The models are available at: http://www.cs.swan.ac.uk/%7Ecsmarkus/ProcessesAndData/Models.

[5] Compared to the given control table, we add `RouteName4` to cover the exit track.

The ERTMS components exchange a number of messages, see Fig. 2. 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 Nat -> Msg .
```

grants a movement authority (encoded as a natural number, determining the position to which the train is allowed to travel) to a specific train with an object identifier of type `Oid`. Messages are urgent, i.e., their processing time is 0:

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

4.2 Instantaneously Reacting Sub-Systems

The processing time of controller, interlocking, and RBC is negligible compared to the time that it takes a train to pass a track. Thus, in our modelling we assume that these three components react instantaneously. In Maude this is expressed by saying that these components do not pose any time constrains. Here, written for the controller:

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

Controller. An ERTMS controller issues route requests. For a general safety analysis, a *random controller* that can make any order of route requests should be considered:

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

However, it is also possible to perform safety analysis relatively to a specific strategy, e.g., a *round-robin controller* that requests routes as follows – 1A first, followed by 1B, until route 4, starting over with 1A again:

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

Yet another parameter are the times at which the controller makes route requests. For both controllers we work with a constant frequency.

Interlocking. In rail control systems, the interlocking provides a safety layer between controller and track. To this end, it monitors the physical rail yard (`occ` says which tracks are currently occupied, `pointPositions` says for each point if it is in normal or in reverse position), manages locks (`pointslocked` says if a point is currently locked by a route), and stores which routes are currently set (`routeset`):

```
class Inter |  routeset : MapRouteName2Bool,
               pointslocked : MapPoint2Bool,
               occ : MapTrack2Bool,
               pointPositions : MapPoint2PointPos  .
```

The interlocking is a passive component, i.e., only upon receiving a message it possibly changes its state and/or sends a message. A typical rule for preserving safety is the following:

```
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 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.

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 on the rail yard: trains use continuous data to represent their position (in our model the distance from the leftmost point of the rail yard); the interlocking uses discrete data (track occupation, set routes, point positions) in its logic. In our model, we take a rather simplified and also abstract view on the challenges involved. We make the assumption that trains request a new movement authority only on the track on which their current authority ends. Furthermore, we abstract the mapping between continuous and discrete data to the two tables presented in Fig. 1.

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,
            designatedRoutes : MapOid2RouteName  .
```

Also, the RBC is a passive system component. A typical reaction is the following: When the interlocking sends a "proceed message" for a route `RN`, the RBC sends a new "end of authority" to the train and removes the corresponding request from its internal state.

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

4.3 Trains

The `Train` class is the only time dependent entity in our model. It is designed as an automaton with four states `stop`, `acc` for accelerating, `cons` for constant speed, and `brake`. There are transitions `stop` → `acc` → `cons` → `brake`, and `acc` → `brake` and vice versa. In addition, it has fields representing the current

distance (relative to a given reference point 0), speed, acceleration, movement authority (relative to 0), maximum speed, and the current track segment.

```
class Train | state : TrainState, dist : NNegRat,
              speed : NNegRat, ac : NNegRat, ma : NNegRat,
              tseg : Track, maxspeed : NNegRat .
```

We assume that acceleration is linear, and – apart from Scenario 3 in Sect. 5.2 – use a value of 1 for both acceleration and deceleration. Trains move according to Newton's laws, i.e., if at time 0 a train is at `DT` with speed `S` and acceleration `A`, then the speed at time `R` is `S + A*R` and the location is `DT + S*R + A*R*R/2`. Its braking distance `bd(S,A)` is `S*S/2*A`. We show the rule for a train in the accelerating state.

```
crl [acc] :
 < O1 : Inter | pointPositions : PointSettings >
 delta(< O : Train | state : acc, dist : DT, speed : S,
         ac : A, ma : MA, tseg : AN, maxspeed : MAX >, R)
      =>
      < O1 : Inter | pointPositions : PointSettings >
      trackseg(PointSettings, < O : Train |
      state : if (S + A * R == MAX)
              then cons
              else (if R == mteMA(DT,S,A,MA)
                    then brake
                    else acc fi) fi,
      dist : DT + S * R + R * R * A * (1/2),
      speed : S + A * R > ) if not AN == Exit .
```

The rule computes the new configuration of a train after time `R` from its old configuration and the interlocking. It is sufficient to list those attributes that are updated, here speed, location, and, possibly, the state. The operator `trackseg` takes the new location of the train and the `PointSettings` from the interlocking and returns a new train object. In the case that the train has entered a new track it will update the train object accordingly. Here, we combine the delta rule together with a state transition, allowing us to exactly determine when a state transition occurs. An alternative approach would be to decouple these orthogonal concepts by expressing the rule as equation + rules. This, in turn, may lead to improvements when model checking.

The time `R` is determined by the maximal time elapse which is, in the acceleration state, the minimum of the following three cases. (1) maximum speed is reached, (2) the end of a track segment is reached, (3) the distance to the movement authority is not greater than the required braking distance.

```
ceq mte (< O : Train | state : acc,  dist : DT, speed : S,
           ac : A, ma : MA , tseg : AN, maxspeed : MAX >)
           = min((MAX monus S) / A,
                 ((endof(AN) + 1) monus DT) / S,
                 mteMA(DT,S,A,MA))  if S > 0 .
eq mteMA(DT, S, A, MA) = (((MA monus 1) monus DT) monus
                          (S * S / (2 * A))) / ( 2 * S) .
```

In case (1) we used `monos` for the maximum of the difference between two numbers and `0`. For cases (2) and (3) the calculation of `mte` involves quadratic equations. From `DT + S*R + A*R*R/2 < endof(AN)+1` we could determine `R` using an approximation via Newton's method. However, since, thanks to our default

tick, we have `0 < R <= 1`, and therefore `0 < A*R*R/2 <= A*R/2`, we approximate the quadratic term either from below or from above depending on the context: in the case of entering a new track we ignore the quadratic term, and put the sampling point slightly late, as we want to be on the new track already; in the case of calculating where to start braking, we bring the event slightly forward, i.e., we start braking slightly too early. Both approximations are justified by the default tick.

4.4 Safety Condition

For classical railway signalling, we established the following finitisation theorem: if a signalling system is collision free for two trains, then it is collision free for any number of trains [9]. 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 are always more than, say, 40 m apart. Thus, we check for the invariant "no collisions":

```
eq { REST  < train1 : Train |  tseg : T1 , dist : N1 >
           < train2 : Train |  tseg : T2 , dist : N2 > }
   |= 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, normal) or T1 == next(T2, reverse) or
         T2 == next(T1, normal) or T2 == next(T1, reverse) ) )
   implies ((N2 monus N1 > 100) or (N1 monus N2 > 100)) .
```

This formula reads: a configuration with two objects `train1` and `train2` of type train models the parameterised formula `nocrashDistance` iff the state of the two trains objects under consideration are in the relation specified after the equal sign. Here, `T1` and `T2` are the tracks and `N1` and `N2` are the positions on which the two trains are respectively. In the formula we check that the trains are more than 100 m apart, provided they are not on the `Entry` or `Exit` track, and provided they are on the same (`T1 == T2`) or on adjacent tracks.

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 Fig. 1, 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. We note that we use the value of 100 m for our invariant. This is different from the desired 40 m, but necessary due to our time sampling strategy: we sample the system only once every second. Within this time, the distance between two trains can reduce by maximally 60 m as we consider trains that travel at a maximum of 60 m/s.

4.5 Completeness

An important question is whether our modelling is complete, that is all errors can be detected by our modelling. Ölveczky and Meseguer give criteria for completeness in object oriented Real-Time Maude [15]. Essentially, one needs to prove that the maximal time elapse function is time robust. This is clearly the case if

we consider movement without acceleration. It is almost all the time the case for our modelling with acceleration, however the small shifts of the sampling points require further analysis. We expect that a weakening of Theorem 4 [15], which takes approximation into account, holds. A necessary premise for this theorem is non-zenoness for which we give the following argument.

Our modelling is non-zeno in the sense of Henzinger [8] as there are no cycles in the behaviour of the automaton which allow time to converge. The argument is that any cycle will involve the accelerating state, which requires a new movement authority to be granted that will extend the current movement authority by at least one. This causes a minimal time elapse bounded away from zero by a fixed amount since the speed of a train is limited.

5 Validation Through Simulation and Error Injection

Here 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 the Real-Time Maude Tool [16] to provide the relevant counter-examples. 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 *Control Tables*, *Release Tables* and *RBC tables*.

5.1 Simulation

We first demonstrate the behaviour of one train moving through the rail yard in Fig. 1 with a start position on track `AA` and a movement authority of 1498. For this we use the Real Time Maude `trew` command to execute our model upto a given time bound.

```
(trew {
  < inter1 : Inter | pointPositions : (P1 |-> normal,
                                       P2 |-> normal) , ... >
  < train1 : Train | state : acc, dist : 2, speed : 0, ac : 1,
                     ma : 1498, tseg : AA , maxspeed : 60 > }
in time <= 39 .)
```

The train accelerates until it begins to brake at the distance of 749.72m:

```
Result ClockedSystem : { < inter1 : Inter | ...>
  < train1 : Train | ac : 1, dist : 1499446241/2000000,
    ma : 1498, maxspeed : 60, speed : 38671/1000,
    state : brake, tseg : AA >} in time 38671/1000
```

A query one time step later shows that a movement authority request is made.

```
{marequest(train1,AA) < inter1 : ...>
 < train1 : Train | speed : 37671/1000, ... >} in time 39671/1000
```

Now, 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 <= 78 .)
```

As no follow-up route is available in the RBC, the train stops at 1497.46 m.

```
  {< inter1 : Inter | ... > < rbc1 : RBC | ... >
   < train1 : Train | dist : 1497446241/1000000, ma : 1498,
     speed : 0, state : stop, tseg : AA >} in time 38671/500
```

To continue, assume that we start in the configuration where the interlocking has set `RouteName3` and the train has made a movement authority request.

```
(trew {marequest(train1,AA)
  < inter1 : Inter | routeset : RouteName3 |-> true,... >
  < train1 : Train | state : brake, dist : 760, speed : 37,
    ac : 1, ma : 1498, tseg : AA , maxspeed : 60 >
  < rbc1 : RBC | availableRoutes : (RouteName3), ...  >
  } in time <= 17 .)
```

Below we see that the authority is extended to 6499 m, and P2 gets locked. Time 17 is when the train crosses to track AB and can accelerate to maximum speed.

```
{ < inter1 : Inter | occ : (AA |-> false, AB |-> true),
  pointslocked : P2 |-> true, ... >
  < rbc1 : RBC | availableRoutes : empty, ... >
  < train1 : Train | dist : 3001/2, ma : 6499, speed : 52,
  state : acc,tseg : AB >} in time 17
```

5.2 Error Injection

We now show that our modelling is able to find errors in the design of various ERTMS components. The following scenarios use our random controller and check the safety condition presented in Sect. 4.4. Furthermore, we model one slow train (max speed 20 m/s) and one fast train (max speed 60 m/s).

```
eq initState = {...
  < train1 : Train | state : stop, dist : 0, speed : 0,
    ac : 1,  ma : 1, tseg : Entry , maxspeed : 20 >
  < train2 : Train | state : stop, dist : 0, speed : 0,
    ac : 1,  ma : 1, tseg : Entry , maxspeed : 60 > ...} .
```

Scenario 1 – Incorrect Control Tables. We consider a scheme plan where the designer forgets to put track section AC into the various interlocking tables in Fig. 1. Model checking highlights that two trains may be within 100 m of each other, with both trains on track AC.

```
{...< train1 : Train | ac : 1, dist : 3249, ma: 3249,
      maxspeed : 20, speed : 0, state : stop, tseg : AC >
    < train2 : Train | ac : 1, dist : 1939979/625, ma : 6499,
      maxspeed : 60, speed : 60, state : cons, tseg : AC > ...}
```

Scenario 2 – Incorrect RBC Tables. We consider a scheme plan where the designer incorrectly calculates an EoA of 3449m for route 1 A in the RBC tables given in Fig. 1. Model checking highlights that two trains may be within 100 meters with `train1` overrunning onto track AD due to the incorrect EoA and `train2` approaching on AC.

```
{...< train1 : Train | ac : 1,dist : 3449,ma : 3449,
      maxspeed : 20,speed : 0,state : stop,tseg : AD >
    < train2 : Train | ac : 1,dist : 12433788921/4000000,
      ma : 6499,maxspeed : 60, speed : 60,state : cons,
      tseg : AC > ...}
```

Scenario 3 – Incorrect Train Braking Parameters. The computation of the braking distance for a train is based on various parameters, some of which may be incorrectly entered by the driver. Hence the train's physical braking distance may differ from the computed one. Below we consider a starting scenario where a deceleration value of 1 (hard-coded, for illustration) has been incorrectly entered for `train2`, whilst the physical train has a deceleration value of 8/10. The other train has correct parameters.

```
{...< train1 : Train | state : stop, dist : 3249, speed : 0,
      ac : 1, ma : 6499, tseg : AD , maxspeed : 20 >
    < train2 : Train | state : stop, dist : 1, speed : 0,
      ac : 8/10, ma : 1, tseg : Entry , maxspeed : 60 > ...}
```

The incorrect parameter causes the two trains both to be on track `AF` within 100 m of each other. This is due to the incorrect behaviour of `train2` which overruns its movement authority thanks to its wrong braking parameter.

```
{...< train1 : Train | ac : 1,dist : 15662341/2500,ma : 6499,
      maxspeed : 20,speed : 20,state : cons,tseg : AF >
    < train2 : Train | ac : 4/5,dist : 968593576867/156250000,
      ma : 7999,maxspeed : 60,speed : 60,state : cons,
      tseg: AF > ...}
```

6 Model Checking Results

In this section we verify a number of rail yards with the Real-Time Maude Tool [16]. We check that the invariant "no collisions", c.f. Sect. 4.4, is globally true, either for all time

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

or for 300 time steps:

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

Here, `initState` is as given in Sect. 5.2. As track plans, we consider the pass-through station shown in Fig. 1 as well as some variations of it, see Fig. 3. This is in order to obtain an indication of how variations in the complexity of the rail yard influence the time required for model checking.

We check all three track plans with manually constructed tables that we consider to be correct. In all settings the model checking confirms that these rail yard designs are collision free (within the given time-bound, if applicable). The table shows verification times[6] and the number of rewrite steps for the three rail yards against the random controller and the round-robin controller (see Sect. 4.2). The following table presents our model checking results (Table 1).

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

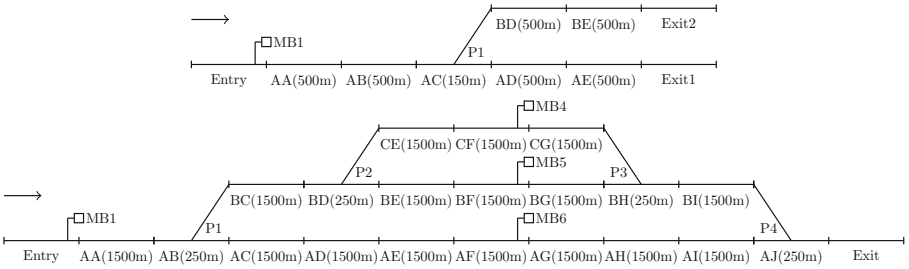


Fig. 3. Track plans for a junction and three platform station.

Table 1. Performance results of model checking three scheme plans.

Scheme Plan	Round Robin Controller Unbounded	Random Controller in Time 300
Junction	0.5 s/1,465,601 rewrites	361.1 s/151,564,627 rewrites
Pass-through Station	0.7 s/1,886,303 rewrites	589.0 s/500,397,040 rewrites
Three Platform Station	1.2 s/2,622,022 rewrites	1957.9 s/1,009,144,410 rewrites

The table shows 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 puts on train movements through the sheme plan. However, when using our random controller, the state space vastly increases. Thus, we provide results for up to a given time bound of $300s$. Note that this time is enough to ensure that two trains can travel completely through the Junction and Station scheme plan. As expected, model checking times increase with the complexity of the scheme plans. It is future work, to consider further, more varied rail yards.

7 Related Work

ERTMS is a complex system of systems, made up of distributed components interconnected through standard (e.g. Euroradio) and proprietary (e.g. Siemens-specific) protocols and algorithms. Our approach reflects this by covering the full control cycle between controller, interlocking, radio-block centre and trains. Our objective is 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.

Our approach to cover all components is different from several verification approaches with a focus on a single component only. Vu et al. [18] provide a generic and re-configurable model of ERTMS Level 2 on the design level sharing our objective. They present their model as a Kripke structure and verify high-level safety properties such as head-to-head collision or derailment on a

point. Their approach abstracts from trains and the RBC and presumes these components to be correctly implemented. Thus, their verification focuses on the interlocking component. Cimatti et al. [4] 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. They focus on software verification of a sub-component rather than on location specific data for the whole system. Nardone et al. [14] develop a new, rail specific specification language DSTM4Rail, an extension of hierarchical state machines. They employ 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. This work is specialised to quality assurance for one ERTMS component.

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 shall comply with a SIL 4 development process.

Chiappini et al. [3] work towards the formalisation and validation of the overall ERTMS/ETCS specifications. 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.

8 Summary and Future Work

In this paper, we have modelled, validated, and verified a complex system of systems of hybrid nature. To this end, we presented an analysis of the ERTMS system, described its information flow, and provided a concise model in Real Time Maude. This model is astonishingly small: it consists of around only 1000 lines of code. We believe this is due to the advanced concepts, especially the object orientated features that Real Time Maude offers. Through simulation we have demonstrated that our model exhibits a number of expected behaviours. Furthermore, by systematic error injection, we have shown that safety in ERTMS depends on all its components. This simulation and error injection give us confidence that our model is valid. Finally, we have presented a number of model checking results that indicate that, for small rail yards, complexity of model checking of physical safety properties is under control.

It is future work to explore further, more complex rail yards, including bi-directional ones. On the practical side we intend to extend our modelling with further controller strategies and more complex train progression behaviour. On the more theoretical side, we plan to investigate completeness and abstraction techniques to reduce model-checking time, including finitisation.

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 helpful advice from Peter Ölveczky on Real-Time Maude and the constructive comments given by three anonymous referees. Finally, we thank Erwin R. Catesbeiana (Jr.) for timely hints on how to stay on track.

References

1. openETCS (2015). http://openetcs.org. Accessed 30 August 2015
2. Alcatel, Alstom, Ansaldo Signal, Bombardier, Invensys Rail and Siemens. System Requirements Specification, Chap. 2, Basic System Description (2006). SUBSET-026-2
3. Chiappini, A., Cimatti, A., Macchi, L., Rebollo, O., Roveri, M., Susi, A., Tonetta, S., Vittorini, B.: Formalization and validation of a subset of the european train control system. In: Proceedings of ICSE 2010. ACM Press (2010)
4. Rizzo, T., Sanseviero, A., Roveri, M., Narasamdya, I., Tchaltsev, A., Lazzaro, A., Corvino, R., Cimatti, A.: Formal verification and validation of ERTMS industrial railway train spacing system. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 378–393. Springer, Heidelberg (2012)
5. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L. (eds.): All About Maude. LNCS, vol. 4350. Springer, Heidelberg (2007)
6. Eker, S., Meseguer, J., Sridharanarayanan, A.: The Maude LTL model checker. In: WRLA 2002, vol. 71, ENTCS. Elsevier (2002)
7. European Railway Industry. ERTMS (2015). http://www.era.europa.eu/Core-Activities/ERTMS/Pages/home.aspx. Accessed 30 August 2015
8. Henzinger, T.A.: The theory of hybrid automata. In: Inan, M.K., Kurshan, R.P. (eds.) Verification of Digital and Hybrid Systems. NATO ASI Series, vol. 170, pp. 265–292. Springer, Heidelberg (2000)
9. James, P., Moller, F., Nga, N.H., Roggenbach, M., Schneider, S.A., Treharne, H.: Techniques for modelling and verifying railway interlockings. STTT **16**(6), 685–711 (2014)
10. James, P., Moller, F., Nguyen, H.N., Roggenbach, M., Schneider, S.A., Treharne, H.: On modelling and verifying railway interlockings: tracking train lengths. Sci. Comput. Program. **96**, 315–336 (2014)
11. James, P., Roggenbach, M.: Encapsulating formal methods within domainspecific languages: a solution for verifying railway scheme plans. Math. Comput. Sci. **8**(1), 11–38 (2014)
12. Lawrence, A., Berger, U., James, P., Roggenbach, M., Seisenberger, M.: Modelling and analysing the european rail traffic management system in Real-Time Maude. In: FTSCS 2014 - Preliminary Proceedings (2014)
13. Meseguer, J., Ölveczky, P.C.: Semantics and pragmatics of Real-Time Maude. Higher-Order Symbolic Comput. **20**(1–2), 161–196 (2007)
14. Nardone, R., Gentile, U., Peron, A., Benerecetti, M., Vittorini, V., Marrone, S., De Guglielmo, R., Mazzocca, N., Velardi, L.: Dynamic state machines for formalizing railway control system specifications. In: Artho, C., Ölveczky, P.C. (eds.) FTSCS 2014. CCIS, vol. 476, pp. 93–109. Springer, Heidelberg (2015)
15. Ölveczky, P.C., Meseguer, J.: Abstraction and completeness for Real-Time Maude. In: WRLA 2006, vol. 176, ENTCS (2007)

16. Meseguer, J., Ölveczky, P.C.: The Real-Time Maude tool. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 332–336. Springer, Heidelberg (2008)
17. Thorvaldsen, S., Ölveczky, P.C.: Formal modeling and analysis of the OGDC wireless sensor network algorithm in Real-Time Maude. In: Bonsangue, M.M., Johnsen, E.B. (eds.) FMOODS 2007. LNCS, vol. 4468, pp. 122–140. Springer, Heidelberg (2007)
18. Vu, L.H., Haxthausen, A.E., Peleska, J.: Formal modeling and verification of interlocking systems featuring sequential release. In: Artho, C., Ölveczky, P.C. (eds.) FTSCS 2014. CCIS, vol. 476, pp. 223–238. Springer, Heidelberg (2015)