A formal design of the Hybrid European Rail Traffic Management System

Paolo Gaspari, Elvinia Riccobene
Università degli Studi di Milano - Italy
elvinia.riccobene@unimi.it

Angelo Gargantini
Università degli Studi di Bergamo - Italy
angelo.gargantini@unibg.it

ABSTRACT

Railway Transportation Management Systems are an emerging field in the context of advanced distributed software systems. Methods and techniques supporting rigorous formal design of system architecture where software components interact with each other and control physical components are highly demanded to assure reliability of the system operation.

We present a formal model of the Hybrid ERTMS/ETCS Level 3, the new standard of the European Rail Traffic Management System, aiming to replace the different national train control and command systems by a unique European railway management system.

We use the Abstract State Machine (ASM) formal method to provide a complete specification of the standard. The model has been developed through a sequence of model refinement steps following the incremental way in which requirements describe train operation. We have exploited the ASMETA tool-set supporting the ASMs to simulate the abstract models and validate them with respect to the operational scenarios that are given as part of the requirements. We discuss ambiguities and inconsistencies of the requirements, as well as difficulties encountered in the specification and during scenarios simulation.

CCS CONCEPTS

• Software and its engineering → Functionality; Formal methods; Software safety.

KEYWORDS

Formal design, Hybrid ERTMS/ETCS, Abstract State Machines

1 INTRODUCTION

The ASMETA (ASM mETAmodeling) framework1 [2] provides a set of tools for ASM model editing, simulation, scenario-based validation, animation, verification, testing, model review, runtime monitoring, etc. Tools are strongly integrated in order to permit reusing information about models during several development phases.

2 HYBRID ERTMS/ETCS LEVEL 3 PROTOCOL

This is a short and introductory explanation of the Hybrid ERTMS Level 3. In the sequel, by the term principles we refer to the requirements of the ERTMS HL3 standard given in [9].

The standard has been proposed to optimize the use and occupation of railways. It proposes the division of the track into separate entities, each named Trackside Train Detection (TTD). In addition, each TTD is subdivided into sub-entities called Virtual Sub-Sections (VSS). A TTD has two possible states: free and occupied with a safety invariant stating that if a train is located on a TTD, then the state of the TTD must be set to occupied. In addition to these two states, a VSS may have the unknown or the ambiguous state. The ambiguous state is used when the information available to the system suggest that two trains are potentially present on the VSS. The unknown state is used when the system can guarantee neither the presence nor the absence of a train on the VSS. For an optimal safety, Movement Authorities (MA) are evaluated and assigned to each connected train. The MA of a train designates a portion of the track on which it is guaranteed to move safely.

The Level 3 of the standard ERTMS foresees an hybrid way to detect train position and train integrity (i.e., a train not accidentally split). All trains are under the supervision of an external unit, the Radio Block Centre (RBC), which knows each train individually and regularly receives, from the train on-board equipment – called Train Integrity Monitoring System (TIMS) –, information regarding train speed, position and integrity (TIMS trains). The hybrid ERTMS Level 3 foresees also an Euroradio-based communication between trains and trackside equipment: trains which are disconnected from the RBC are no longer lost, since they are still visible by means of this trackside train detection (TTD) system (ERTMS trains).

ERTMS Level 3 Hybrid principle accommodates, therefore, different types of trains including ERTMS trains equipped with the Train Integrity Monitoring System (TIMS), ERTMS trains without TIMS, and non-ERTMS trains.

The main document we have referred to for modeling the railways control system is the standard in [9]. However, since requirements are ambiguous and incomplete in some points, very fragmented, difficult to understand for someone that is not a domain

1http://asmeta.sourceforge.net/
We consider trains moving along the track and operating under the way in which the VSS status must be updated is simply an outcome according to the standard, each VSS has to behave accordingly to this state machine is the arrival point of our modeling.

RBC would work also if more than one track controller based on its position reports.

3 MODELING PROCESS

We consider trains moving along the track and operating under the control of the RBC and the Track Controller. Therefore, principles of the ERTMS HL3 are captured by the parallel execution of a number of ASM agents: the Trains, all behaving similarly, the RBC and the Track Controller. Code 1 reports the main rule (i.e., the starting point of model computation) of the multi-agent ASM.

Each of these agents has very specific tasks, modeled by means of ASM transition rules:

- the train has different behavior according to its status: integer, disconnected, reconnecting, lost integrity; mute timer), and trains data collection;
- the track controller handles: train chased/chasing, shadow train, two train in one VSS, train on TTD, Movement Authority of chasing train, ghost train.

All the agents contribute to change of the status of the VSS. According to the standard, each VSS has to behave accordingly to a state machine (reported in Fig. 7 at pag. 24 of the principles and on the right side of Fig. 1) specifying the change of the VSS status. This state machine is the arrival point of our modeling.

Fig. 1 depicts the idea of our multi-agents ASM behavior, where the agents trains, RBC and Track controller operate in parallel to change the status of the VSS, and the state machine describing the way in which the VSS status must be updated is simply an outcome of our specification.

Each agent’s program has been refined, step by step, starting from some basic operation till to get the specification of all agent’s tasks. For example, train behavior is being refined till to capture the most complex configuration of a train that can be integer, can be connected/disconnected, can lose integrity, can be chased/chasing, can be shadow, can be involved in the sweeping of VSS sections.

To understand and specify the agents’ behavior, we started from the textual description of the principles, and we expressed the text in terms of transition rules, by supplying the necessary definitions of domains and functions. In naming functions/domains, we use an application-oriented language that can be understood by the stakeholders. An example is shown in Code. 2 that reports the requirements of train reconnecting in section 3.8.2 of the principles, and the corresponding ASM rule.

The complete ASM model has been developed by the following sequence of eight model refinement levels:

L0 The first model refers to the operation of an integer train, abstracting from the interaction with the RBC and the Track Controller (integer_train.asm);
L1 We refine the model at L0 by adding the specification of an integer train interacting with the RBC (ermts_hl3_integer_train.asm);
L2 This level introduces the train operation in case it loses connection to the RBC (ermts_hl3_integer_disconnected_train.asm);
L3 This level considers an integer and/or disconnected train that can lose integrity and change its length, and that operates by interacting with the RBC; we also model the Track Controller to deal with chased/chasing train status (ermts_hl3_integer_disconnected_lengthchanged.asm);
L4 This level adds to the previous model the specification of train operation in case of reconnection to the tracksde (ermts_hl3_integer_disconnected_lengthchanged_reco.asm);
L5 This level refines the model by adding the specification of the sweeping mechanism (ermts_hl3_integer_disconnected_lengthchanged_reco_sweep.asm);
L6 This level introduces the use of the timers and control of two trains in a VSS (ermts_hl3_integer_disconnected_lengthchanged_reco_sweep.timers.asm);
L7 This level includes all the refinements needed to execute the scenarios. They have been simulated starting from models at level L6. In most of the cases, refinements consisted in specific function/domain instantiation to represent the train configuration according to the scenario description. In other cases, refinements were due to handle information and behavior not considered in the requirements.

During modeling, we needed to introduce some concepts not directly stated in the principles. For example, to manage situations as (a) the train is on the conjunction of two subsequent VSS, (b) the train is split into two trains, (c) estimate the min safe rear end of the train and express other parameters (as train speed, breaking factor, etc.), we do not consider a VSS an atomic unit, but we think it measured in VSS_Units.

Our models, one for each refinement level, are available on-line at https://sourceforge.net/p/asmeta/code/HEAD/tree/asm_examples/examples/ermts_etcs/}

\footnote{In the ASM-[

\section*{3 MODELING PROCESS}
3.8.2.1 Train Reconnecting

When a train reconnects after the “mute timer” has expired, the VSS sections set “unknown” when the “mute timer” expired can be restored based on the following conditions: (1) the VSS sections where the train is located will become “occupied” if the train reports “integrity confirmed”, no change of train data train length was reported since the previous position report and there is no shadow train risk. If these conditions are not fulfilled, these VSS sections will become “ambiguous”. (2) ...

macro rule r_Trains_reconnecting($train in Trains, $vss in Vss) =
   if not mute_timer($train) and vss_states($vss) = UNKNOWN and
   contains(train_state($train), NOTCONNECTED)
   then
   par
       if (trains_confirmed_integrity($train) and not(length_changed($train))
       and not(trains_shadow($train)))
       then r_vss_train_occupied[$train]
       else r_vss_train_ambiguous[$train]
   end

Code 2: Translating text into rules

agent Trains r_Trains_on_TrackSide[]
macro rule r_Trains_on_TrackSide =
   par
   forall $position in Vss_Units
   with $position <= second(ma(self)) and $position => first(ma(self))
   do
       r_Trains_Run[self, vss related_to_train_position($position)]
       r_collect_train_data[self]
endpar

Code 3: Train program

At each level of abstraction, model execution and scenarios simulation have been a fundamental means to test how to refine the model and to better understand the principles (see discussions in Sect 4). We do not report here the complete specification, but we provide some excerpts of our models and some hints in order to be able to read, understand and simulate the entire chain of models available on-line. Models have been developed and simulated by mean of the ASMETA framework.

In the following sections we mainly focus on presenting the structure of agents’ programs in terms of macro call rules, and the program refinement. Further details (signature and transition rules) can be found on-line.

3.1 Operation of trains
A train operation is specified by the rule r_Trains_on_TrackSide (the Trains agent’s program) shown in Code 3. A train moves on the track (rule r_Trains_Run) based on its Movement Authority (MA), i.e., the permission to run to a specific final location (within the constraints of the infrastructure). MA is specified by the monitored function ma: Trains -> Prod(Vss_Units, Vss_Units) that returns, for a train, the pair (starting_position, final_position).

At each step, in order to deal with specific situations (see Sect 4), a train stores two subsequent (current and previous) configurations. Therefore, while moving, the train stores its current vital data (i.e., position, length, speed, and MA, given as monitored values), and save its previous configuration. The rule r_Trains_Run is refined along the chain of refined models. At level L0, it models train operation, abstracting from the interaction with the RBC and the Track Controller. As shown in Code 4, the rule simply calls the macro r_Trains_integer that specifies the requirements for an integer train operation given in section 3.5 of [9].

At the refinement level L1, rule r_Trains_Run is refined by adding the rule r_Trains_connected modeling the operation of a connected train, and so complementing the integer train operation. At this level, the rule r_Trains_Run models the train behavior in its basic configuration, i.e., a train able to:
- calculate the minimum safety distance from a chasing train;
- confirm its integrity if it was connected also in the previous state;
- manage his mute_timer⁶;
- move along the track.

At the refinement level L2, the r_Trains_disconnection rule is added to the rule r_Trains_Run to model the disconnected train status. The rule r_Trains_disconnection allows managing all conditions described in sections 3.8.1 and 4.2.1 of the requirements, i.e., if the train disconnects, then
- the state of the VSS occupied by the train becomes UNKNOWN;
- all VSS that are part of the MA of the train become UNKNOWN;
- the VSS ahead of the train that are part of its MA become UNKNOWN until the first free TTD;
- when the disconnect propagation timer expires onto adjacent VSS, the UNKNOWN state must be propagated, backward and forward, up to the first free TTD, or up to the VSS occupied by the train, or up to the VSS of the MA of the chasing train on the previous TTD.

At the refinement level L3, we include, to the train movement (r_Trains_Run), the rule r_Trains_lostintegrity concerning the loss of integrity; it models requirements in section 3.7. The train that has lost integrity continues to travel on the track, and the VSS that the train leaves become UNKNOWN. Subsequently, the VSS will be set to FREE when the entire TTD (the VSS are part of) becomes free. Particular attention must be paid to a chasing train having MA that ends in the TTD where the chased train loses integrity. When a chased train loses integrity, or communicates a

⁶This task is shared with the RBC till level L6. At level L7, scenario validation showed the necessity to refine the model and specify only the RBC as the agent responsible of the mute timer management.

Code 4: Integer train operation

Code 5: Train movement at level L5
change in length, or the wait integrity timer expires, the state of the VSS occupied by the train becomes AMBIGUOUS.

At the refinement level L4, rule r_Trains_reconnecting was added to model the train reconnection (section 3.8.2). This rule fires if the train is in an UNKNOWN VSS and its mute timer expires; it manages the following cases:

- if the train confirms integrity, it does not report length change w.r.t. the previous state, and there is no risk of shadow trains, then the VSS state becomes OCCUPIED, otherwise it becomes AMBIGUOUS;
- if the train still has the initial value for starting point of the MA, then the VSS ahead of the train position becomes FREE;
- if the train confirms its integrity, there is no length change compared to the previous state, and there is no other train in the VSS occupied by the train, then the VSS behind the train becomes FREE.

At the refinement level L5, the rule r_Vss_Sweeping is added to model the sweeping mechanism of the VSS (section 3.9.1). If a train had received permission to enter a VSS in state UNKNOWN, it is in a VSS in state OCCUPIED and there is no risk of shadow trains, then, when the train leaves the VSS, this VSS becomes FREE. To model this situation that requires to process a past event, we introduce a presweeping state for a VSS in order to save the necessary information. Code 5 reports rule r_Trains_Run.

At the refinement level L7, all rules called by r_Trains_Run are slightly refined to manage differences come out by comparing what is expressed in the scenarios and what is required in the principles. We discuss these issues in Sect. 4.

3.2 RBC operation

Introducing the agent RBC (Radio Block Centre) was necessary to model concepts and events not directly attributable to train behavior, but contributing to change the VSS status. Such information was initially left abstract and modeled by mean of monitored functions. However, too many constraints would have been necessary on these functions to guarantee correct runs. So we decided to model the RBC and delegate it the control of some train parameters.

Agent RBC is introduced at the refinement level L2. Here the agent’s program (r_Rbc_Supervisor) consists of the rule r_check_rbc_trains_connection that controls some parameters, such as the status of a train (connected or not), the confirmation of train integrity, the mute timer.

At level L3, the RBC program is extended by the rule r_check_train_status, used to model the train status communication to the RBC. By train status we mean a series of fundamental information for the behavior of the whole system, such as loss of train integrity, change of train length, status of the train connection with the RBC, being a shadow train. No additional features (i.e., rules) are added to the agent program at levels L3, L4, and L5. At the L6 refinement level we deal with the management of the timers. Some types of timers are not directly connected to the train, but to other entities; e.g., the integrity loss propagation timer refers to the VSS, the ghost propagation timer and the shadow train timer refer to the TTD. At level L6, the following rules are added to refine the RBC program: r_check_ghost_propagation_timer, r_check_sweeping_timer, r_check_shadow_trains, r_check_ghost_trains, and r_check_rbc_trains_connection_collect_data. Code 6 reports the complete RBC agent’s program.

3.3 Track controller

During modeling, we realized that it was appropriate to introduce an additional agent having the task of handling the interactions between two trains during their movement on the track. Indeed, trains, in addition to having their own state (e.g., connected, not integer, length-chanced), contribute to determine some properties of the other trains on the track (for example, a shadow train is a non-connected train that is behind a connected train), and, in turn, contribute to determine the state change of the VSS. Agent Track Controller appears at level L2, when the rule r_check_rbc_trains_connection is introduced. For each train t, this rule monitors trains chasing t, according to their relative distance given by the function confirmed_safe_rear_end.

At the refinement level L4, the rule r_check_two_reporting_trains_connection_collect_data is added to check if two trains are on the same VSS (i.e., it checks whether the tail of the forward train occupies the same VSS of the train head behind it), and sets the VSS status accordingly.

At the refinement level L6, checks are introduced to determine whether a train is a ghost or a shadow train (a ghost train is not

---

**Code 6: RBC operation**

```java
// create an Rbc and set its program
agent Rbc: r_Rbc_Supervisor[]
macro rule r_Rbc_Supervisor =
  par
  r_check_train_status[]
  r_check_ghost_propagation_timer[]
  r_check_integrityloss_propagation_timer[]
  r_check_trains_chasing_chased[]
  r_check_trains_chasing_chased[]
  r_check_ghost_trains[]
  r_check_shadow_trains[]
  r_check_rbc_trains_connection_collect_data[]
endpar
```

**Code 7: Track Controller operation**

```java
// create a Track Controller and set its program
agent TrackController: r_TrackController[]
macro rule r_TrackController =
  par
  r_check_two_reporting_trains_inonevss[]
  r_check_two_reporting_trains_inonevss[]
endpar
```
connected and is unknown at the platform, a shadow train is a ghost train that follows a connected train). VSS status is updated on the base of the train status by means of the rules r_check_ghost_trains and r_check_shadow_trains that specify requirements 4.2.2 and 4.5, respectively. An extra rule, r_check_ma_of_chasing_trains, is assigned to the agent Track Controller to limit the MA of a chasing train moving on a VSS in state UNKNOWN (sect. 4.2.1.6).

At the last level of refinement L7, the Track Controller’s program is refined to add checking of trains on a TTD. In case of negative feedback, the Track Controller updates the TTD(s) status to FREE. This task is performed by the r_check_trains_onttd rule. For the sake of completeness, it must be said that, at level L7, rules in the Track Controller’s program are slightly refined to handle some ambiguities emerged during scenarios simulation (see Sect.4).

Code 7 reports the complete Track Controller’s program.

4 SCENARIOS VALIDATION

Scenarios of case study described in [9] were simulated at level L6; however, some simulations motivated a further refinement level (L7) as already discussed.

The movement of trains along the track takes place by instantiating, for each train, those monitored specifications using the following attributes: (1) the Movement Authority (MA), provided as starting and ending positions expressed in terms of Vss_Units; (2) the current Train Position, expressed in terms of Vss_Units; (3) the current Train Speed, necessary to calculate the min safe rear end; (4) the Trains Length, expressed in terms of Vss_Units; (5) the Train status, a sequence of Boolean variables, based on the scenario to be simulated, describing the status of the train. Details on scenarios simulation can be found in the material we make available on-line. For each scenario we provide: (1) a table (file ERTMS3-Scenarios.xls) reporting values of the monitored variables at each step of the simulation, and some notes regarding differences, problems, and unclear situations between description of the scenarios in [9] and model simulation; (2) a graphical representation of the scenario execution: we add a thumb up in case of successful validation or an alert symbol when scenario simulation differs from scenario description, and some warnings are reported; (3) the ASM run of the scenario (Code 8 is an excerpt of file trace_scenario1.asm); (4) the ASM run of the scenario (Code 8 is an excerpt of file trace_scenario1.asm); (5) the ASM run of the scenario (Code 8 is an excerpt of file trace_scenario1.asm).

For further documentation, some of the scenarios have been validated by exploiting the scenario-based validation [4], available as part of the ASMETA framework. The approach allows to build a precise scenario against which the developer wants to validate the model, by setting given values of monitored functions and checking the updated values of precise locations upon execution of one or more steps of the ASM. Code 9 reports an example of scenario construction for (a part of) the operation scenario 1 in [9].

Scenario starts with a configuration where Train 1 of length 10 moves along a track that is made of 3 TTD joint at positions 10, 20 and 30, and of 8 VSS (from Vss_11 to Vss_33). RBC and TrackController are active. Integrity is confirmed (length_changed(train1) is false) and the train is connected (lost_integrity(train1) is false). Train 1 receives, as Movement Authority, to move to final position 60 with a speed of 200. Upon one step, Train 1 in position 10, moves with speed 200 to the final position 60 and its length has not changed (values are given by the function collect_trains_data(train1)).

All the 9 scenarios have been simulated and, apart from scenario 8 that aroused some unclear and inconsistent situation between principles and scenario description, we have been able to reproduce their description in [9], although, in some cases, scenario description and simulation are not completely aligned. This does not represent a limitation of our models, but it is due to the following reasons:
We present the most recent work that takes into consideration requirements (so that in \[7\]) and validation by simulation, leaving verification as future work. Of trains, we have preferred to concentrate on model specification (readable than plain Event-B), which are easier for domain experts to understand and validate. Many of these contributions use the model checker ProB to find invariant violations with counterexamples \[10\], and such counterexamples were used to validate scenarios \[11\]. However, model checking has a bottleneck in the number of model variables, and this was a problem for a large model as that of this case study. Furthermore, a part of the work in [7] that uses animation, in the other contributions it is difficult to check how the model can reproduce the informal scenarios. Some of these contributions \([7, 10, 11]\) guarantee safety property on the movement of trains. We have preferred to concentrate on model specification and validation by simulation, leaving verification as future work.

Among other approaches, in \[3\], the Spin model checker is used for modeling and validating the ERMTS HL3. The authors prove a set of safety properties regarding the correct movement of trains, and simulate the nine scenarios reported in the requirements. While modeling, the authors abstracted away many unnecessary requirement details and needed to handle with the expressive limitation of Promela, the input language of the Spin model checker. Although the approach is more suitable for verification, such Promela models are usually less readable than notations as ASMs and B.

In \[5\], a formal model for the ERMTS HL3 is presented by using Electrum, a lightweight state-based specification language that extends Alloy. The Analyzer component of Electrum is used to perform scenario exploration to validate the model. To achieve these results, the authors needed to make a number of abstractions. As us, they abstract continuous aspects of the rail traffic management domain due to the discrete nature of Electrum.

## 6 CONCLUSIONS

We have presented the ASM specification of the ERMTS HL3 standard as originally provided by the ERTMS Users Group. The process of requirement specification went through a sequence of ASM refined models, and model refinement was the only way to handle the complexity of the requirements and to develop a formal model able to simulate all the proposed scenarios. Tool-based model validation helped to reveal ambiguities and inconsistencies. Although deeper verification analysis must be performed to prove refinement correctness and application domain properties, we believe our specification can be considered a rigorous support to discuss incompleteness and ambiguities of the requirements and a formal base for reliable implementation.

## REFERENCES


\[11\] http://www.cs.swan.ac.uk/~csmarkus/ETWG-RC/