Diagram-led formal modelling using iUML-B for Hybrid ERTMS Level 3

D. Dghaym[https://orcid.org/0000-0002-2196-2749], M. Poppleton, and C. Snook[https://orcid.org/0000-0002-0210-0983]

ECS, University of Southampton, U.K.
`{dd4g12,mrp,cfs}@ecs.soton.ac.uk`

Abstract. We demonstrate diagrammatic Event-B formal modelling of a hybrid, 'fixed virtual block' approach to train movement control for the emerging *European Rail Traffic Management System* (ERTMS) level 3. We perform a refinement-based formal development and verification of the no-collision safety requirement. The development reveals limitations in the specification and identifies assumptions on the environment. We reflect on our team-based approach to finding useful modelling abstractions and demonstrate a systematic modelling method using the UML-like state and class diagrams of iUML-B. We suggest enhancements to the existing iUML-B method that would have benefitted this development.

1 Introduction

Railway control systems are safety-critical, and it is common for railway safety standards (e.g. CENELEC EN-50126, EN-50128/9) to recommend the use of formal modelling and verification to certify their correctness. We present our application of a diagrammatic formal modelling method to such a system.

The *European Rail Traffic Management System* (ERTMS)[1] [6] will comprise a single ATP (automatic train protection) system and a single GSM radio communication system train-to-trackside, to replace the variety of current national train control solutions. Hybrid ERTMS Level 3 [8] - a compromise between full ERTMS Levels 2 and 3 - aims to increase network capacity at reduced cost, using existing trackside train detection equipment together with radio communication.

This case study concerns a physical environment of trains, and communication by radio and trackside equipment. The case study concerns the control of trains moving on a linear track which is part of a wider network controlled by an interlocking system which is out of scope of this case study. A train movement controller called the *Radio Block Centre* (RBC) manages the *Movement Authority* (MA) granted to each train in mission. The focus of this work, called the *Virtual Block Detector* (VBD), conservatively estimates train locations to a finer granularity than physically detected track sections, and thus reports free virtual track sub-sections available for train movement. Trains and trackside report location data to the VBD. In turn the VBD reports free track sections to

[1] http://ertms.net

RBC. The MA granted to each train consists of a set of sections that the train is permitted to move into. A *controlled* train is instructed that the MA sections are free; a *trusted* train is instructed that they might not be free. The key safety property which we verify is that controlled trains do not run into trains that are ahead of them.

Motivation and Contribution The case study presents challenges - addressed by our contribution - for a formal development method, typical of challenges arising in safety-critical cyber-physical systems. First is the development of a useful model reflecting the component architecture of the target system (VBD) interacting with its physical environment and other system components. The model enables us to verify functional safety properties of the specification. Second, we need readable models so that domain experts are able to validate the model. While we do not focus on validation in this paper (other than for our own sanity checks of the model), future work will include running scenarios in a form of model acceptance test. Third, we describe how we tackle the diffcult process of turning a detailed and complex specification that contains ambiguity and relies on tacit domain knowledge, into a formally precise model containing useful abstractions. (We view this contribution as particularly useful for industrial partners to enable them to adopt formal modelling techniques). Fourth, we have the emergent critique of the specification document: assumptions on the environment, omissions, ambiguities, errors etc.

The refinement-based Event-B modelling method [1] is an appropriate choice since it allows us to verify key properties while leaving certain features, and interacting components, abstract and underspecified. The architecture can be layered through the refinement: each layer can focus on an abstract component interface, the environment, or a specific feature of the target system. Event-B has strong tool support [2] for verification and validation in the form of theorem provers and model-checkers. Diagrammatic modelling notations and tools are available which help in conceptual modelling: we use iUML-B class diagrams and state-machines [17,16,14]. One of our goals is to show that using iUML-B leads to a readable formal specification (or at least more readable than plain Event-B), which is easier for domain experts to validate.

Structure The paper is structured as follows. We next recall Event-B and iUML-B basics in sec. 2. Sec. 3 reviews our development process, and sec. 4 gives our system analysis. The refinement strategy is then summarised (sec. 5), followed by a detailed account of modelling in sec. 6. Next is related work (sec. 7), followed by the conclusion in sec. 8.

2 Event-B and iUMLB

Event-B [1,9] is a refinement-based formal method for system development. An Event-B model contains two parts: *contexts* for static data, and *machines* for dynamic behaviour specified by *variables* `v`, *invariant* predicates `I(v)` that constrain the variables, and *events*. An event comprises a guard denoting its enabling-condition and an action describing how the variables are modified when the event is executed. In general, an event `e` has the following form, where `t` are the event parameters, `G(t, v)` is the guard of the event, and `v := E(t, v)` is the action of the event.

```
e == any t where G(t,v) then v := E(t,v) end
```

Event-B is supported by the *Rodin platform* (Rodin) [2], an extensible toolkit which includes facilities for modelling, verifying the consistency of models using theorem proving and model checking techniques, and validating models with simulation-based approaches.

iUML-B [14,16,17] provides a diagrammatic modelling notation for Event-B in the form of state-machines and class-diagrams. The diagrammatic elements share the repository of an Event-B model, and contribute to that model. For example a state-machine will automatically generate the Event-B data elements (sets, constants, axioms, variables, and invariants) to implement the states, and contribute additional guards and actions to existing events. Class diagrams provide a way to visually model data relationships. Classes, attributes and associations are linked to Event-B data elements (carrier sets, constants, or variables) and generate constraints on those elements.

3 Process

Formal models are often presented as if they were developed in perfect inexorable steps when, in practice, they never are. We give an overview of our informal team-based process illustrating the iterations that involved many misunderstandings failures and re-work. Although we had some feedback from domain experts on terminology and detailed clarifications, this was not substantial as we wanted the case study to test our ability to use formal methods to understand and interpret the specification. The domain experts were not involved in the process described in this section. The team consisted of research and academic staff who had some experience of formal modelling of railway applications such as interlockings and crossings, but no previous experience of communications-based, virtual section train control.

Systems Analysis While the *Hybrid ERTMS Level 3* (HLIII) specification is quite well presented in terms of explanatory scenarios, its focus makes it a detailed requirements specification for the VBD. It does not explain the overall system aims and principles so well. We therefore started by reverse engineering our understanding of the system in order to understand its purpose and the concepts that it is based on. This involved analysis of the information in the specification, discussions and sketching whiteboard diagrams such as components, entity relationship and state-machine diagrams. The diagram-based analysis naturally led into the iUML-B modelling. The systems analysis identifies the main components in the system and the information flow between them. This is necessary for the model to reflect the appropriate responsibilities of the VBD verses assumptions it makes upon other components. As with most stages of the modelling process, the analysis was iterative. The modelling helped our understanding of the system and our new understanding helped us choose better abstractions for modelling. For example initially we assumed that only connected trains were 'in-mission'. However, when modelling we realised that when a connection is lost the system relies on the fact that the train will continue to respect its MA and this implies that the train is still in-mission. This new understanding of the system led us to revise our models so that the in-mission state-machine was independent of (i.e. parallel with) the connected state-machine.

Refinement Strategy The refinement strategy provides a plan for how we intend to build the model, choosing abstractions, adding details in refinement steps and introducing invariant properties at appropriate stages. We considered two alternative approaches, a) start from an abstract safe system or b) start from an unsafe system and make it safe. For this example we chose the second approach. While the first approach is perhaps more traditional, in this case, the safety properties were not so obvious and were complicated by unsafe, albeit mitigated, scenarios. So we wanted to capture the essence of train movement before introducing assumptions and progressing towards details that can distinguish between safe scenarios and mitigated unsafe scenarios. Again, the refinement strategy evolved as we discovered difficulties and adapted our approach.

Modelling In modelling we used iUML-B for its diagrammatic notation which follows on from the diagrams used in our analysis and review stages. As usual, we used the provers to verify models and when they fail, and we cannot be sure why, the ProB model checker helps to find counter examples. We also animated the models to check that the model behaves as expected.

Review We held regular reviews to discuss problems with the modelling. As indicated in the previous steps, the reviews led to significant iterations to our understanding of the system, revisions to our refinement plan and consequent changes to the model. Problems fell into the following categories:

- We cannot prove this PO - look for a better modelling approach. Example: contiguity of *next_VSS* relationship - We found it difficult to prove contiguity properties about *Virtual Sub-Section* (VSS) using abstract properties. While this should, in principle, be possible, we decided it was not worth the effort and introduced numeric indexing of VSS (relying on the contiguity of a range of integers). We retained the *next* function for elegance of expression in guards and actions.
- This is not a useful refinement - change refinement strategy. Example: We wished to introduce features such as timers as soon as it was possible to do so (i.e. when the triggering functionality was available). However, we had not yet introduced the relevant VSS state changes to utilise the timeout. To rectify this we altered our refinement strategy to introduce abstract versions of VSS states and associated transitions.

– This is not a true data refinement - change systems analysis. Example: As we modelled the flow of information through the control components we found it difficult to reconcile the reported train positions and controlled MA with the safety properties of the abstract environment. It seemed that we would need to introduce some form of responsiveness assumptions to limit the difference between actual and control variables. However, the specification implied that the VSS states were asynchronously updated. As our understanding of the MA principle improved we realised that the position inaccuracy is of no consequence and we adjusted our systems description.

4 System Analysis

The HLIII specification is a detailed description of one component (the VBD) of a wider system that controls train movements. The other components involved in the system are the trains and trackside equipment, which we refer to as *environment* (ENV), and the RBC that calculates movement authorities limiting the movement of trains.

The VBD receives messages from trains and train detectors. It also receives information about the output of the RBC. It calculates a set of sections that it believes to be free of any trains and sends these to the RBC. The RBC sends to each train, a movement authority consisting of a set of sections that the train may move into. The train is either instructed that the sections are all free or that they might not be free. We wish to model and verify item 3, the VBD. To do this we also need to consider (and model) the other 2 items.

The **environment** consists of a linear track divided into fixed sections (*Virtual Sub-Section* (VSS)) with trains moving in one direction on the track. Detectors (*Trackside Train Detection* (TTD)) report when a train is present. However, there is only one TTD for a group of VSS. There are 2 kinds of trains; those that communicate with the control system, and those that do not. Trains that communicate send three items of information to the VBD:

– their current position (in finer granularity than track sections),
– the length of the train,
– whether the train is confirmed as integral.

Communicating trains are able to receive information about the range of sections they are allowed to move through and whether the authorised track is guaranteed to be free (full-supervision) or not (on-sight). For the purpose of this description we partition trains into three kinds: ghost trains (not communicating), controlled trains (communicating with guaranteed free sections authorised) and trusted[2] trains (communicating with possible non-free sections authorised). Trains that do not communicate can only be detected by TTD and may move

[2] *Controlled* and *trusted* (trains) are terms that we have introduced, they are not terms from the specification.

freely according to some assumptions concerning physical limitations and those imposed by train design regulations.

The **RBC** grants movement authority (permissions) to the communicating trains. The RBC uses information it receives from the VBD about which VSS are free. An MA consists of a set of track sections that the train is allowed to move through. The train is also instructed as to whether it needs to be responsible for avoiding collisions with trains in front (*On-Sight Movement Authority* (OSMA)) or whether it can assume the track sections are free (*Full Supervision Movement Authority* (FSMA)). We assume the RBC always issues safe FSMA in accordance with the information it receives from the VBD. I.e all sections in an FSMA are ones that the VBD has calculated to be free.

The **VBD** is responsible for deciding which VSS are free based on information it receives from the TTD and from *Positive Train Detection* (PTD) communications received from communicating trains. It sends information about which VSS it believes are free to the RBC. Since PTD reports may be intermittent or interrupted and some trains do not communicate at all, the estimate of free VSS is cautious in these circumstances.

The positions of trains that are communicating are known fairly accurately (subject to some lag in communications) from the PTD data sent by the train (position, length and integrity) as well as physical limits on possible train movement in between communications. The position of the train may cover a range of sections from that occupied by the rear to that occupied by the front. Some robustness is necessary to accommodate limitations of the communication mechanisms such as temporary loss of communication etc.

The position of a train that is not communicating (i.e. a ghost train) is difficult to determine. The possible positions of a ghost train are estimated as a range of sections based on the following:

- its last known position (from a PTD or a loss of integrity),
- how far it could possibly have travelled since its position was known,
- information from trains and free TTD that delimits its movement range.

A ghost train is created in the VBD by one of the following means: a communicating train stops communicating, a TTD spontaneously and unexpectedly detects a train, or a communicating train reports that it has lost integrity.

For loss of integrity, a ghost train is created just behind the communicating train to represent the detached section of carriages. A communicating train is converted to a ghost train if the train's mute timer expires (after communication is lost) or if it sends a mission end message and terminates communication. A ghost train is removed (i.e. destroyed) by sweeping. Sweeping is the movement of a trusted train (with OSMA) through the sections where the ghost train may be. If the trusted train is able to pass through the sections the ghost train does not exist. A ghost train may also be converted to a communicating train if it starts communicating with the VBD (either by sending a mission start communication or by re-starting previous communication).

5 Refinement Strategy

Through system analysis and iterative modelling, the original outline refinement strategy evolved into the following. The target VBD model interacts with the physical environment of trains and trackside: the first refinement layers ENV. Next is the RBC component, followed by lower layers which elaborate the VBD.

ENV-M-1 Trains: Defines a linked list of trains to keep track of train order and prevent overtaking. Trains are created at the rear of the linked list and removed from its front. We also allow adding a new train in the middle of the linked list as a result of train split.

ENV-M0 Train movement, VSS: Introduces the train movement in terms of VSS section updates, where a VSS section is either free or occupied by a train. The train movement is modelled as an independent update of the position of the train front and rear.

ENV-M1 Ghost vs connected trains: Distinction between connected and ghost (i.e., non-connected) trains, where all new trains join as ghost.

ENV-M2 TTD: Introduces TTD sections which can be either free (no train on any of its VSS) or occupied (a train on at least one of its VSS). The TTD state is immediately updated by train movement events.

RBC-M3 RBC: RBC can grant trains MA. We call trains with MA inMission, where the RBC may extend or shrink their MA while connected.

VBD-M4 Position reporting: Presents the VSS four states (free, occupied, ambiguous, unknown). Also introduces the reported versus actual train position with the associated MA trimming. Disconnection related timers are also introduced.

VBD-M5 Controlled vs trusted trains: Fully supervised FS (controlled) vs on-sight OS (trusted) trains are introduced. An OS train has unsafe MA and is assumed not to crash into the back of other trains. An FS train has safe MA and therefore cannot crash into the back of other trains. In addition to Ghost train timers.

VBD-M6 Integrity loss If a train reports either integrity loss or changed length, the train is split. Additionally, integrity loss propagation timers to control availability of adjacent VSS are introduced.

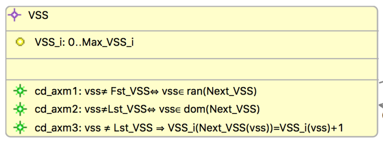
VBD-M7 Lower levels Full VSS state transition as per specification, including all timers.

6 Modelling

The model consists mainly of two parts: the ENV and the VBD. The RBC provides an intermediate layer for moving from the ENV to the VBD.

Modelling the Environment In the first part of the model, we focus on modelling the ENV and the possible trackside events, such as train movement, splitting and loss of communication.

In the context, we model the network topology using iUML-B class diagrams (Fig. 1). First we introduce the *TRAIN* class (not shown in figures), then the *VSS* with their linear layout enforced by indexing via attribute, *VSS_i*, Fig. 1a.


VSS
VSS_i: 0..Max_VSS_i
cd_axm1: vss≠ Fst_VSS⇔ vss∈ ran(Next_VSS)
cd_axm2: vss≠Lst_VSS⇔ vss∈ dom(Next_VSS)
cd_axm3: vss ≠ Lst_VSS ⇒ VSS_i(Next_VSS(vss))=VSS_i(vss)+1
0..1
Next_VSS
0..1
(a) VSS Properties

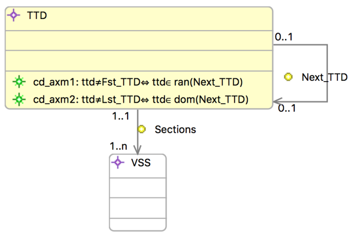

TTD
cd_axm1: ttd≠Fst_TTD⇔ ttd∈ ran(Next_TTD)
cd_axm2: ttd≠Lst_TTD⇔ ttd∈ dom(Next_TTD)
0..1
Next_TTD
0..1
1..1
Sections
1..n
VSS
(b) TTD Properties

Fig. 1: Class Diagram representing the track in the context

At the abstract level, we introduce how trains can join and leave the network or in other words how trains can be created and destroyed. The variable class *train*, with superset *TRAIN*, represents the trains that currently exist in the network. There are two cases for creating trains, either a train can join from the beginning of the network or in the middle as a result of splitting behind an existing train. An important property at this level is: *trains cannot overtake*, which is why we introduce the relative ordering of the trains, represented by the variable association *next_train* in Fig. 2. Therefore, a train can only leave the network if there is no train in advance, this is represented by the guard $tr \notin dom(next\_train)$ added to the method ENV_leave_network of class *train*.

In the next refinement, we model train movement. A train's position is given by the VSS that it occupies: variable association *occupiedBy* in Fig. 2. We only model trains moving forward, hence a train can only leave a VSS if it occupies the next one. In order to ensure the no overtaking property, a train can only move forward if it doesn't share a VSS with its next train. Apart from splitting, a train can only join the network from the first VSS and trains can only leave from the last VSS. Since the no-overtaking property is fundamental to the safety of the system, we ensure the model does not break it by introducing the following invariant, which states that a train cannot occupy a vss with an index higher than the lowest indexed VSS of the next train:

$$\forall tr1, tr2 \cdot (tr2 \mapsto tr1) \in next\_train \implies$$
$$max(VSS\_i[occupiedBy \sim [\{tr2\}]]) \leq min(VSS\_i[occupiedBy \sim [\{tr1\}]])$$

To distinguish between trains that are communicating and those that are not, we introduce sub-states *connected* and *ghost*, of *train* (Fig.3).

Next, we introduce the *TTD* which group sets of contiguous *VSS* via association *Sections* (Fig. 1b). Class *occupiedTTD*, which is a sub-class of *TTD*, represents those TTD that have at least one of their VSS occupied by a train. At this level, we distinguish two cases when a train is leaving the last VSS of

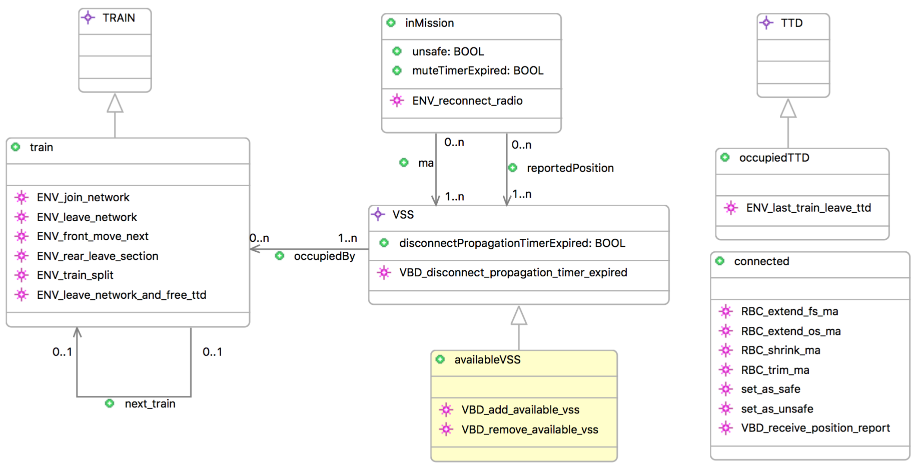


Fig. 2: Class diagram representing dynamic aspects of the environment

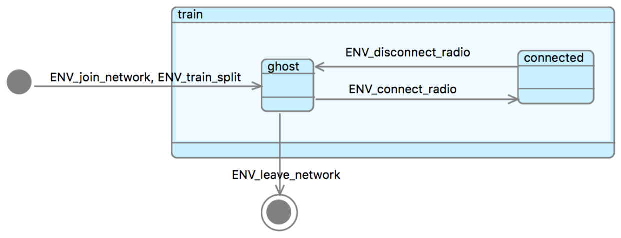


Fig. 3: Train communication statemachine

the TTD: i) no other train occupies the TTD and the TTD becomes free (and is removed from *occupiedTTD*) or ii) it remains occupied and not free. The same applies to a train leaving the network which can also free a TTD.

In the final environment model, we introduce the RBC role which paves the way for the VBD part. The RBC provides movement authorities (MA) which we assume trains will respect. The MA is modelled as a variable association *ma* between *train* and *VSS*. We refine the train statemachine further by introducing a parallel state-machine (Fig. 4). The sub-states, *inMission* and *noMission*, distinguish the mission status of trains. *inMission* represents trains that have performed a *Start of Mission* (SoM) (transition ENV_start_of_mission), while *noMission* represents trains that either did not start or performed an *End of Mission* (EoM) (transition ENV_end_of_mission). The mission state-machine was introduced as a parallel state-machine to the communication state-machine so that trains that lose communication retain their mission status. All connected trains have a mission. This is ensured by the invariant: $connected \subseteq inMission$

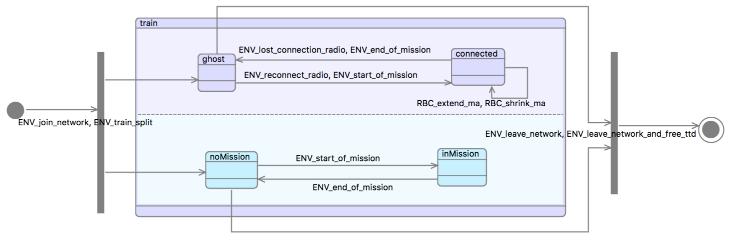

train
ghost
connected
ENV_lost_connection_radio, ENV_end_of_mission
ENV_reconnect_radio, ENV_start_of_mission
RBC_extend_ma, RBC_shrink_ma
ENV_join_network, ENV_train_split
noMission
inMission
ENV_start_of_mission
ENV_end_of_mission
ENV_leave_network, ENV_leave_network_and_free_ttd

Fig. 4: Parallel statemachines for communication and movement authority

We also split the radio connection/disconnection transitions in Fig. 3 into two cases to distinguish between SoM and reconnection and connection loss and EoM. The transitions ENV_start_of_mission and ENV_end_of_mission are common to the two statemachines. Note that when a train first joins a network, it joins as a ghost train with no mission, and when leaving the network it also has to leave as a ghost train with no mission.

When a train performs SoM, it is immediately granted an MA for the VSS it occupies. However, this does not allow the train to move to new VSS sections. In order to move forward, the RBC should extend the MA as shown by the self transition RBC_extend_ma of the *connected* state in Fig. 4. Our assumption that trains with a mission respect their MA is enforced by the *inMission* class invariant: $occupiedBy \sim [\{tr\}] \subseteq ma[\{tr\}]$ [3]. However, when the RBC shrinks the *ma* (e.g. due to propagation of an unknown VSS state) the actual train position may have progressed sufficiently to violate this invariant. We believe this is a limitation of the system and therefore introduce a boolean attribute *unsafe* of class *train* to indicate that the train has entered an unsafe scenario (to be detailed in later refinements), and the invariant can be violated in this scenario: $occupiedBy \sim [\{tr\}] \subseteq ma[\{tr\}] \vee unsafe(tr) = TRUE$. In Fig. 2, RBC_trim_ma in the *connected* class plays the role of a garbage collector, removing the VSS the train has left behind.

Modelling the VBD The VBD cannot see directly what is happening in the ENV; it depends on periodic reports (PTD) sent by the train and it then asynchronously updates the VSS states. Similarly, the RBC receives information about VSS state from the VBD. This asynchronous behaviour relies on the fact that the actual position cannot be behind the reported position and is somewhere within the MA. I.e. reported position is only used to free VSS after a train has passed. This is embodied in the following invariants of class *inMission* which relate the actual position *occupiedBy* with the *reportedPosition* seen by the VBD.
$min(VSS\_i[reportedPosition[\{tr\}]]) \leq min(VSS\_i[occupiedBy \sim [\{tr\}]])$

[3] Note that class invariants are implicitly quantified over instances of the class, hence the antecedent $\forall tr \cdot tr \epsilon inMission$ is added automatically

$max(VSS\_i[reportedPosition[\{tr\}]]) \leq max(VSS\_i[occupiedBy \sim [\{tr\}]])$
We also refine loss of connection with mute timer expiry. We model time abstractly without introducing a clock, giving timeouts a non-deterministic opportunity to expire. When a mute timer expires this will enable the disconnect propagation timer whose expiry will affect the VSS state in later refinements.

In the next refinement of the VBD, we distinguish between the two different modes of MA: FSMA and OSMA. In FSMA mode the RBC only uses free VSS to extend *ma*. In OSMA mode, the RBC can extend *ma* with any VSS since we trust the OSMA trains not to crash. This behaviour is modelled in Fig. 5 by partitioning *inMission* into two different sub-states, *controlled* and *trusted* representing FSMA and OSMA modes respectively. The choice between the two transitions, RBC_extend_os_ma and RBC_extend_fs_ma, is non-deterministic and determines the mode of the train.

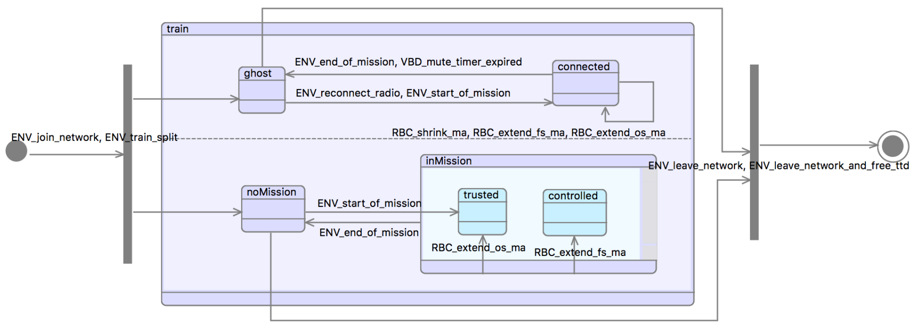

train
ghost
connected
ENV_end_of_mission, VBD_mute_timer_expired
ENV_reconnect_radio, ENV_start_of_mission
RBC_shrink_ma, RBC_extend_fs_ma, RBC_extend_os_ma
ENV_join_network, ENV_train_split
inMission
trusted
controlled
noMission
ENV_start_of_mission
ENV_end_of_mission
RBC_extend_os_ma
RBC_extend_fs_ma
ENV_leave_network, ENV_leave_network_and_free_ttd

Fig. 5: Introducing sub-states to represent FSMA and OSMA modes

We can now introduce a safety invariant concerning the separation of controlled trains; the *ma* of controlled trains do not overlap:
$\forall tr1, tr2 \cdot tr1 \in controlled \land tr2 \in controlled \setminus \{tr1\} \implies ma[\{tr1\}] \cap ma[\{tr2\}] = \varnothing$

Hence, the RBC can only extend the *ma* of *controlled* trains using VSS sections that are free and not part of any *ma*. We introduce a sub-class *availableVSS* of *VSS* to represent the free vss sections. This will be refined to the VSS state *free* in future refinements as we introduce the state-machine of the specification. However, extending the *ma* for trusted trains does not have these restrictions.

At this level we add and remove *availableVSS* abstractly with the general conditions that apply for all cases. Therefore for adding a new *availableVSS*, it either belongs to a free TTD or no train has reported its position in this VSS, while for now the only condition for removing a VSS from *availableVSS* is that it belongs to an occupied TTD.

Next we introduce the concept of train integrity. We partition *connected* into two sub-states: *integral* and *nonIntegral*. We also refine the PTD position reports to include integrity information. Therefore, we split the method

VBD_receive_position_report into different cases for confirming integrity, integrity loss, integrity not available and train length change. We also introduce the integrity waiting and propagation timers.

At this stage, it became clearer to us that *availableVSS* is insufficient, and it would have been better to introduce different sub-states of VSS as soon as we started the VBD part. Most timers result in a change to state *unknown* when they expire, hence there is no great benefit from having the timers without showing their effect. Moreover, this would have the advantage of introducing the different transitions of the VSS statemachine earlier, with the four states (*free*, *unknown*, *ambiguous*, *occupied*) and gradually building towards the specification. Such decision requires a new iteration of the Event-B modelling in accordance with the refinement strategy described in sect. 5.

Proof Statistics The current modelling approach (before the next iteration in accordance with sect. 5) contains 8 machines. Our modelling resulted in 246 proof obligations, where about 66% (162) were proved automatically with the default Rodin prover configuration. However, most of the proof obligations that were not discharged automatically were related to well-definedness of min/max operators. We then changed our Rodin configuration to include SMT solvers, this increased the number of automatically discharged proofs to 226 (92%). Finally, we added the relevance filter (but excluding newPP), which is a meta prover that improves the efficiency of the predicate prover by selecting relevant theorems. This improved our automatic percentage to about 99%. However, when recalculating auto-status, we found that this sometimes dropped to 97%. Presumable this is due to fluctuations in the processor resources available to the prover. We managed to get the auto-status back to 99% by increasing the timeout limits of the provers. This high percentage of automation depends on the modelling style applied. For example, we used indexing to avoid abstract models of sequences whose transitive properties are difficult to prove. In our models, we used iUML-B class diagrams and state-machines. The iUML-B state-machines plugin provides two alternative translations, one representing the states as an enumerated set and the other representing states as subsets of the statemachine instances. We used the latter translation, lifting the state-machine to a set of instances (*train*). Therefore, the generated state-machine type invariants are based on subsets of the instance set (*train*). In future work we will assess whether the use of iUML-B and the choice of state-machine translation affect the degree of automatic proof.

7 Related Work

Various approaches have been made during the development of the Event-B method, to integrating it into the broader Software Engineering process. The original interpretation of UML class diagrams and statemachines in classical B [17] have been presented - and tool-supported - as iUML-B [16] for Event-B. More recently Event-B refinement has been extended [14] to this diagrammatic modelling method. Example applications - of which this work is one - include

[11]. CODA [3] is a tool-supported framework extending iUML-B for component-based embedded systems.

Train control is a familiar domain for Formal Methods, and specifically for B and Event-B-based approaches. Butler et al [4] give a methodical treatment of the diagrammatic modelling of the rail interlocking system Railground with both iUML-B and Event Refinement Structures [15]. In [7], the authors present the Event-B development of a *Communications-based Train Control* (CBTC) system from Hitachi Ltd. Their focus is on the use of *Abstract Data Types* (ADTs) to manage the complexity of modelling a graph-based rail network and its dynamics. This example is comparable to ERTMS Level 3 and uses moving blocks. The authors further proposed [10] the extension of iUML-B to support diagrammatic modelling of ADTs, using the same Railground case study as [4].

Other related work such as [13] on Hybrid ERTMS Level 3 is based on moving blocks. These models are hybrid, being concerned with continuous modelling of exact train position and speed reporting. This ABZ2018 case study is the first formal examination of fixed virtual blocks that we are aware of.

8 Conclusion

The specification is a rich and detailed source of information but is written as a functional specification of the VBD component rather than a systems requirements document. While trying to formalise and abstract a model of the system, we discovered several ambiguities. For example, when modelling the mute and disconnect propagation timers we found that section 3.4.2.2 describes the start event of the disconnect propagation timer to be expiry of the mute timer. However, in scenario 4, EoM also starts the disconnect propagation timer. One possible explanation is that the mute timer also operates for trains when they perform EoM. On the other hand, transition 7A in the VSS statemachine distinguishes between mute timer expiry and EoM, implying that these are two different cases. In addition, section 3.4.2.2 states that the mute timer is stopped once the train re-connects, but doesn't describe the EoM case. Does the VBD need to keep a history of train positions with ended mission? Or is the mute timer not stopped in this case? This example illustrates how formal modelling can reveal ambiguities in the specification. Collaboration and interaction with domain experts is crucial to resolve such questions as it would be dangerous to model our own assumptions.

Formal modelling and the need to make abstractions and refine them, helped to develop our understanding of the system and to gain insights into the principles of the design. The main example of this is the link between: *what it does* - prevents certain kinds of collisions; *how it does it* - allocates movement authorities; *why it works* - movement authorities cannot be entered by another train. It also made us very aware of limitiations to the safety of the system such as the case where carriages could roll backwards which could break the *why it works.*

We intend to continue developing and improving the formal model as part of the Enable-S3 project[4]. The model will form a demonstrator for the Rail use case and will be used in conjunction with MoMuT for test case generation [12]. An acceptance test specification will be developed using 'Cucumber for iUML-B' [5] which is a formalised notation for describing test scenarios for iUML-B models. The acceptance tests provide a rigorous, repeatable validation accessible to domain-experts with limited formal methods expertise.

Some suggestions for improvements to the iUML-B notation and tools arose from modelling the HLIII. For example, it is often convenient to initialise class attributes/associations and have a complete mapping of their instances to values rather than specify a common value for each instance. Similarly, we often needed to specify 'class-wide' invariants in which case the Event-B generator adds an unnecessary universal instance quantifier. These improvements will be incorporated in a future release.

Classes represent a set of instances with state represented by attributes and associations and behaviour described in methods. Lifted statemachines represent a set of instances with state represented by the statemachine state and behaviour described in transitions. It is often useful to use both visualisations for the same set of instances. While the diagrams can be linked to the same set of instances, the integration is not very strong and the tooling sometimes conflicts in Event-B generation. We experienced difficulties for example when modelling *connected* (trains) as both a class and a state. A first improvement would be to allow statemachines to be placed inside classes (an existing feature request) and rectify the problems with generation. However, a more fundamental integration might be possible: a common underlying record-based notation for the iUML-B model. In this case the diagrams would be alternative views of a common model. A text representation of the record-based model could also be provided. This would align well with our plans to provide a text based version of iUML-B to improve team-based development (where model diff and merge are essential).

All data supporting this study are openly available from the University of Southampton repository at http://doi.org/10.5258/SOTON/D0403

Acknowledgements
The authors would like to thank Tomas Fischer of Thales Austria GmbH, for his helpful comments about the paper.
This work has been conducted within the ENABLE-S3 project that has received funding from the ECSEL Joint Undertaking under Grant Agreement no. 692455. This Joint Undertaking receives support from the European Union's HORIZON 2020 research and innovation programme and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia, Norway.

References

1. J-R. Abrial. *Modeling in Event-B: System and Software Engineering.* Cambridge University Press, 2010.

[4] https://www.enable-s3.eu/

2. J-R. Abrial, M. Butler, S. Hallerstede, T. Hoang, F. Mehta, and L. Voisin. Rodin: An open toolset for modelling and reasoning in Event-B. *Software Tools for Technology Transfer*, 12(6):447–466, 2010.
3. M. Butler, J. Colley, A. Edmunds, C. Snook, N. Evans, N. Grant, and H. Marshall. Modelling and refinement in CODA. In *Refine@IFM 2013, Turku, Finland, 2013.*, volume 115 of *EPTCS*, pages 36–51, 2013.
4. M. Butler, D. Dghaym, T. Fischer, T. Hoang, K. Reichl, C. Snook, and P. Tummeltshammer. Formal modelling techniques for efficient development of railway control products. In *RSSRail 2017, Pistoia, Italy, 2017*, volume 10598 of *LNCS*, pages 71–86. Springer, 2017.
5. T. Fischer, C. Snook, and T. Hoang. Formal model validation through acceptance tests. Technical report, University of Southampton, UK, March 2018.
6. N. Furness, H. van Houten, L. Arenas, and M. Bartholomeus. ERTMS Level 3: the game-changer. *IRSE News*, 232, 2017.
7. A. Fürst, T. S. Hoang, D. Basin, N. Sato, and K. Miyazaki. Large-scale system development using Abstract Data Types and refinement. *Sci. Comput. Program.*, 131:59–75, 2016.
8. EEIG ERTMS Users Group. *Principles: Hybrid ERTMS/ETCS Level 3.* http://www.southampton.ac.uk/assets/sharepoint/groupsite/Academic/ABZ-Coneference-2018/Public%20Documents/ABZ2018/16E0421A_HL3.pdf. Accessed 18/1/2018.
9. T. Hoang. An introduction to the Event-B modelling method. In *Industrial Deployment of System Engineering Methods*, pages 211–236. Springer-Verlag, 2013.
10. T. Hoang, C. Snook, D. Dghaym, and M. Butler. Class-diagrams for Abstract Data Types. In *ICTAC 2017, Hanoi, Vietnam, 2017, Proceedings*, volume 10580 of *LNCS*, pages 100–117. Springer, 2017.
11. T. Hoang, C. Snook, L. Ladenberger, and M. Butler. Validating the requirements and design of a hemodialysis machine using iUML-B, BMotion Studio, and co-simulation. In *ABZ 2016, Linz, Austria, 2016, Proceedings*, volume 9675 of *LNCS*, pages 360–375. Springer, 2016.
12. W. Krenn, R. Schlick, S. Tiran, B. Aichernig, E. Jobstl, and H. Brandl. Momut::uml model-based mutation testing for uml. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–8, 2015.
13. A. Platzer and J-D. Quesel. European Train Control System: A case study in formal verification. In *ICFEM 2009, Rio de Janeiro, Brazil, 2009. Proceedings*, volume 5885 of *LNCS*, pages 246–265. Springer, 2009.
14. M. Said, M. Butler, and C. Snook. A method of refinement in UML-B. *Softw. Syst. Model.*, 14(4):1557–1580, 2015.
15. A. Salehi, M. Butler, and A. Rezazadeh. Language and tool support for event refinement structures in Event-B. *Formal Asp. Comput.*, 27(3):499–523, 2015.
16. C. Snook. iUML-B statemachines. In *Proceedings of the Rodin Workshop 2014*, pages 29–30, Toulouse, France, 2014. http://eprints.soton.ac.uk/365301/.
17. C. Snook and M. Butler. UML-B: Formal modeling and design aided by UML. *ACM Trans. Softw. Eng. Methodol.*, 15(1):92–122, 2006.