An iterative formal model-driven approach to railway systems validation

Asfand Yar, Akram Idani, Yves Ledru, Simon Collart-Dutilleul, Amel Mammar, German Vega

To cite this version:

Asfand Yar, Akram Idani, Yves Ledru, Simon Collart-Dutilleul, Amel Mammar, et al.. An iterative formal model-driven approach to railway systems validation. 28th International Conference on Engineering of Complex Computer Systems (ICECCS), Jun 2024, Limassol (Chypre), Cyprus. pp.272 - 289, 10.1007/978-3-031-66456-4_15 . hal-04731687

HAL Id: hal-04731687
https://hal.univ-grenoble-alpes.fr/hal-04731687v1

Submitted on 11 Oct 2024

An Iterative Formal Model-Driven Approach to Railway Systems Validation

Asfand Yar[1], Akram Idani[1](✉), Yves Ledru[1], Simon Collart-Dutilleul[2], Amel Mammar[3], and German Vega[1]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, 38000 Grenoble, France
{asfand.yar,akram.idani,yves.ledru,german.vega}@univ-grenoble-alpes.fr
[2] COSYS-ESTAS, Univ. Gustave Eiffel, IFSTTAR, Univ. de Lille, 59650 Villeneuve d'Ascq, France
simon.collart-dutilleul@univ-eiffel.fr
[3] TELECOM SudParis, SAMOVAR, CNRS, Institut Polytechnique de Paris, 91011 Evry, France
amel.mammar@telecom-sudparis.eu

Abstract. European Rail Traffic Management System (ERTMS) is a standard for the train control and signalling system whose application is spreading throughout Europe. The ETCS (European Train Control System) level 3 is attracting experts because it is still in the design phase. Many works provide formal models to the verification of ERTMS/ETCS using formal methods, but they did not investigate the validation problem. To deal with this challenge we propose an iterative formal model-driven approach that helps validating step-by-step a real formal specification of ERTMS/ETCS hybrid level 3. Our approach introduces Domain-Specific Languages (DSLs) to help system experts understand existing specifications that are already proven. To this purpose we extend and apply Meeduse, the only existing language workbench today that allows embedding formal semantics within DSLs. The paper presents this application and discusses the lessons learned from the railway and formal method experts' points of view.

Keywords: Domain-specific languages · B Method · Validation · Refinement · ERTMS/ETCS

1 Introduction

In a train system, functions are distributed and performed by the train or by track-side devices. Introducing new technology can lead to reallocating a function from the track to the train. This is typically the case for the railway ERTMS/ETCS[1] level 3 solution [1], where positions that were detected by the track are now provided by the train using its own localization means.

[1] ERTMS/ETCS: European Rail Traffic Management System/ European Train Control System.

G. Bai et al. (Eds.): ICECCS 2024, LNCS 14784, pp. 1–18, 2024.
https://doi.org/10.1007/978-3-031-66456-4_15

The ERTMS/ETCS has gained significant attention in industry and academia, and because of the underlying safety requirements the formal methods community has investigated numerous techniques to its modeling and verification. For instance, the call of solutions of the ABZ'2018 conference [2] has resulted in several realistic applications of formal methods. Most of these applications deal with verification concerns, without providing insights showing whether their formal model is valid and conforms to user requirements. Some approaches propose translations from graphical models (*e.g.,* UML, KAOS) into formal specifications, but as the transformation is not proven correct, the resulting formal models still need to be validated. And, even if the transformation is proven, validation is still necessary. Indeed, Verification cannot replace validation, and vice versa.

More generally, the use of a model provides an abstraction of reality, assuming a specific point of view [3]. However, in this abstraction, the presence of each concept and correlated requirement has to be justified. While building a Domain Specific Language (DSL) that can be executed, Meeduse [6,7] provides a solution to confront the understanding of the system architect and of the domain expert. This motivates the current work, which is an application of the tool to a real case study. Meeduse is a language workbench built on the B method allowing one to formally define the semantics of DSLs, and animate them using the ProB model-checker. In [15], we presented an approach for visual animation of B specifications using DSLs. In this paper, we extend our approach with an iterative model-driven technique. Considering an existing B specification of an ERTMS hybrid level 3 system provided by a formal methods expert [12], the paper proposes to build incrementally a meta-model of the system which defines the abstract syntax of a DSL. At each level of abstraction, the system is analyzed and the corresponding requirements are simulated. Links are built from the current incremental stage of the model and the corresponding provided B component. This simulation allows railway experts to validate the formal models and compare their understandings.

Section 2 discusses the state-of-the-art and describes shortly the case study of ERTMS hybrid level 3. Section 3 explains our iterative approach. Section 4 presents the approach based on the first level of our DSL. Section 5 discusses other refinement levels and provides the lessons learned from this work. It also talks about a new proposed solution to reduce proof obligations. Finally, Sect. 6 draws the conclusions of this paper and discusses its perspectives.

2 Problem Positioning

2.1 Related Works

Taking into account the railway signalling industrial context, the CENELEC 50128[2] norm should be considered. This norm strongly recommends the use of formal methods for critical software development. Among various formal languages that are used in the railway scientific literature, most of the contributions come from the B ecosystem [5]. Though Formal Methods (FMs) prove

[2] https://standards.globalspec.com/std/2023439/afnor-nf-en-50128.

a railway system, they do not guarantee its conformity to requirements. The misunderstanding of the user requirements by FM experts may lead to wrong systems. Hence, validation is required to inspect whether specifications meet the user requirements. If looking closer to the CENELEC 50129[3] requirements (dealing with the safety of electronic devices used for signalling), a graphical description of the system, structured specification and formal or semi-formal specification are highly recommended for subsystems having the highest Safety Integrity Level (namely SIL3 and SIL4). A structured specification in the context of CENELEC 50129 means "*design hierarchically broken down and fully traceable back to requirements specification*". The ERTMS hybrid level 3 specification which is analyzed in our use case is related to the signalling task. Considering CENELEC 50129, this paper provides DSLs together with their visual animation of the ERTMS B specification, ensuring that specifications meet the user requirements.

In the literature, there are tools providing graphical animation and visualization of B specifications such as BRAMA [13], AnimB[4] B-Motion Studio [8], B-Motion Web [9], VisB [14], and animation function in [11]. However, typical drawbacks are associated with these tools: they use scripting or programming languages for visualization or for mapping. This added programming layer is error-prone. The diagnosis becomes more difficult because errors may come from the added programming layer used for visualization. Another limitation of the tools mentioned above is dealing with the visualization of B specifications with multiple components like the ones we are using in this paper. In the existing tools, the formal method expert has to define visualizations for each component but in our approach, (s)he is only responsible for producing mappings in B. We believe that our approach is less error-prone than other tools, as in our approach, mappings are produced by formal method experts themselves. Also, our technique allows domain experts to provide scalable domain representations by themselves rather than defined by formal method experts.

2.2 A Formal B Model of ERTMS Hybrid Level 3 Use Case

ERTMS/ETCS is a set of specifications introduced as a standard for a common inter-operable platform for railway management and signalling systems. It is intended to be adopted in all European countries and to replace the signalling system with a common system in order to reduce cost and provide interoperability. ERTMS/ETCS is segregated into three levels depending on the used equipment and the operating modes. The first two levels are operational while level 3 is still in the design and standardization phase.

In the literature, one can find numerous formal models of ERTMS/ETCS. In this paper, we focus on the one by Mammar *et* al. [12] using Event-B. In their ERTMS/ETCS 3 model, a railway track (a line to run a train) is divided into sections known as Trackside Train Detection (abbreviated as TTD). A TTD is

[3] https://standards.globalspec.com/std/10280790/dsf-fpren-50129.

[4] AnimB: https://wiki.event-b.org/index.php/AnimB.

further divided into subsections called Virtual Sub-Section (VSS). An ERTMS train can be fitted with TIMS which stands for Train Integrity Monitoring System reporting its position and integrity to the train supervisor. The latter refers to the system controller. ERTMS trains without TIMS can only report their front position while non-ERTMS trains do not report their position at all to the supervisor. A train's VSS occupation is also determined by the TIMS. In the ERTMS/ETCS, Movement Authority (MA) is the permission assigned to the train to move on specific sections or subsections. In this model, the supervisor assigns the MA (containing VSSs) and sends it to the ERTMS train. A train cannot go beyond the specified VSS in order to avoid collisions.

This model allows the trains to be connected or disconnected. A connected train regularly reports its integrity and position to the supervisor. In order to manage disconnected trains, the model provides the concept of timers. Note that, in this use case, all trains move in the same direction on the same track and MA is chosen non-deterministically. We re-use this model in our approach as a classical-B artifact which consists of four components: an abstract machine (M0) and 3 refinements (M1, M2 and M3). This case study represents a linear route of a railway topology with tracks without any branching points (switches).

3 Towards an Iterative Formal Model-Driven Approach

3.1 Meeduse Language Workbench

This work extends and applies the Meeduse[5] language workbench [7]. The tool is dedicated to formally instrument DSLs using the B-Method. It embeds the ProB [10] model-checker to support animation and verification. The Meeduse approach (taken from [6]) is shown in Fig. 1.

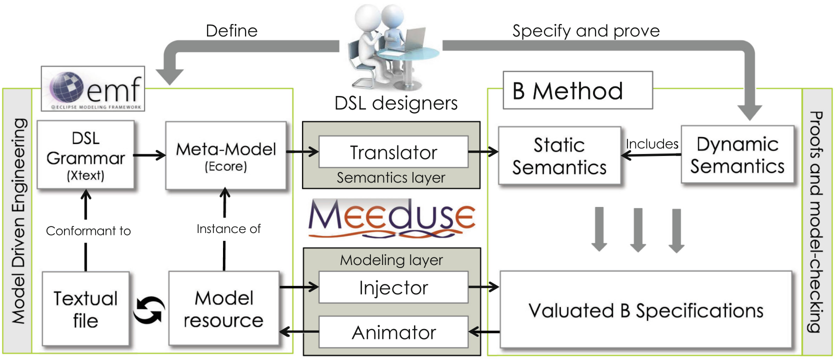


Fig. 1. Meeduse Approach (taken from [6])

The semantics layer translates the meta-model of a DSL into a B specification using the translator component. The resulting B specification defines the static

[5] http://vasco.imag.fr/tools/meeduse/.

semantics of the DSL. The Modelling layer includes two components: Injector and Animator. From a given instance (model resource) of the DSL, Meeduse first creates a valuated B specification which is an extraction from the translated B model, and then it injects the valuations to populate the various B data structures. Having this valuated B specification, ProB is applied to animate B operations of any B machine that includes the valuated model. It is called "Dynamic Semantics" because it confers to the DSL a behavioral character (note that the user specifies the dynamic semantics). At every animation step, when ProB modifies the internal state of the valuated functional model, Meeduse translates back this modification to the input model resource, resulting in an automatic animation of the model resource.

3.2 Using an Existing B Model

In [15], we presented an approach that uses DSLs to visually animate of a given B specification. The approach introduces a linkage machine that allows the usage of the B machine as dynamic semantics of the DSL. This approach is extended here and applied to a realistic case study. We incrementally build our DSL layer and use refinements and inclusions to make the connection between every increment of the DSL and the considered B model. The proposed approach is applied to each abstract/refinement component as depicted in Fig. 2.

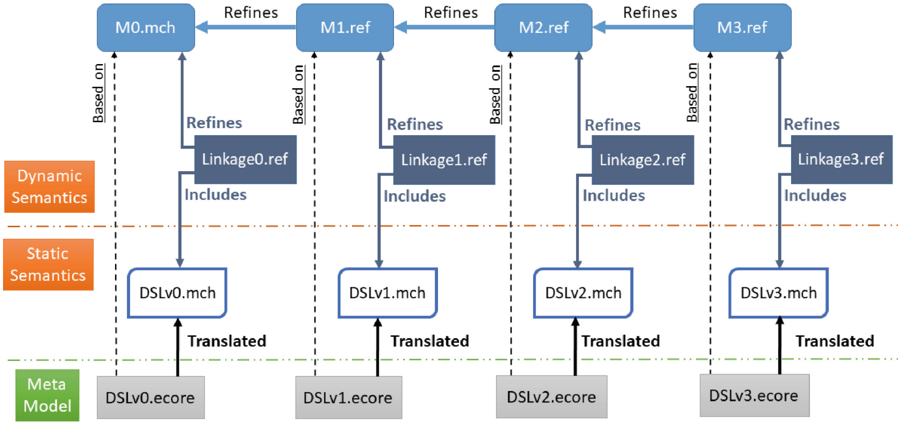


Fig. 2. The Iterative Architecture for the Case Study

First, we develop a DSL meta-model (DSLv0.ecore) based on the initial abstract component (M0.mch) of the model. Then we provide a linkage machine (Linkage0.ref) which refines the M0.mch and includes the translated static semantics of DSLv0.mch (the translation from ecore to B is done using Meeduse). In the next iteration, we update the DSL based on refinement M1.ref (which is a refinement of the M0.mch machine) and the resulting DSL becomes

DSLv1.ecore. In this iteration, another linkage machine (Linkage1.ref) is introduced which refines the existing refinement M1.ref and includes the translated static semantics DSLv1.mch. The same step is repeated until the final refinement M3. At each iteration, we are able to visually animate the existing component using the corresponding version of the DSL thanks to the execution of the linkage machine in Meeduse. This graphical animation allows the domain expert to check that the verified built specification (existing model) captures the right requirements. The complete B specification of the existing model, the ones generated from the DSL and linkage machines can be found in the Meeduse git repository[6]. The mechanisms of linking B data structures, the initializations and the operations of linkage machines can be also found in our paper [15]. The linkage machines in this case study are generated using our DSL-based Linkage Generator Tool[7], which is built on the definition of generation patterns that are defined by the user and that can be applied (and reused) for various specifications and models.

4 An ERTMS/ETCS Hybrid Level 3 DSL

In a Model-Driven Architecture, a DSL is built from a meta-model. We propose to incrementally create this meta-model based on the existing formal B model, such that each concept in the meta-model corresponds to a concept in the B model. From each version of the meta-model Meeduse generates B static semantics, and the dynamic semantics refers to the corresponding component of the B model. Figure 8 (presented later) shows the whole meta-model where concepts of each refinement are defined using different colors.

4.1 DSL Version 0 (DSLv0)

DSLv0 is built based on abstract machine M0.mch of the existing model. Figure 3 shows the B data structure of M0.mch with its sets, constants and variables. Initialization and operations are not shown here due to space limitation. Machine M0.mch allows free movement of trains on TTDs and collisions are possible at this level. This level contains operations: trainSupervisor, trainEntering, trainMovingInSameTTD, trainMovingFrontNextTTD, trainMovingRearNextTTD, trainExiting, trainConnect, trainDisconnect, and TimerExpiration.

Figure 4 shows the meta-model of DSLv0 where class Railway is the root class. It is composed of class Trackside and class Train. Class Trackside has the attribute TrackStatus which can have the value Free or Occupied from the enumeration Status. Each Trackside has 0 to 2 previous tracksides (association previous) and 0 to 2 next tracksides (association next).

Class Train has two attributes in addition to its identifier: kindOfTrain and Connected. Attribute kindOfTrain refers to three kinds of trains (enumeration

[6] https://github.com/meeduse/Samples/tree/main/ETCSLevel3.

[7] https://github.com/meeduse/Samples/tree/main/LinkageGeneratorTool.

```
MACHINE
    M0
SETS
    StateTTD =  {freeT, occupiedT};
    TRAINS ;
    TrainKind = {TimErtms, Ertms, NoErtms}
CONSTANTS
    Ttds, minTTD, maxTTD, trainKind, Trains, Cars
VARIABLES
    stateTTD, trainOccupationTTDFront, trainOccupationTTDRear,
    isConnected, supervisor
INVARIANT
  stateTTD ∈ Ttds → StateTTD
   ∧ trainOccupationTTDFront ∈ TRAINS ⇸ Ttds
   ∧ trainOccupationTTDRear ∈ dom(trainOccupationTTDFront) → Ttds
   ∧ ∀ tr . (tr ∈ dom(trainOccupationTTDFront)
        ⇒ trainOccupationTTDRear(tr) ≤ trainOccupationTTDFront(tr))
   ∧ isConnected ∈ trainKind ⁻¹ [{Ertms, TimErtms}] → BOOL
   ∧ supervisor ∈ BOOL
```

Fig. 3. B data structure of existing abstract machine M0

KindOfTrains): ERTMS, NoERTMS and TIMSERTMS. An TIMSERTMS train is equipped with Train Integrity Management System contrary to simple ERTMS trains. Recall that ERTMS trains equipped with TIMS communicate their location with the supervisor. Attribute Connected is a Boolean that shows the connection of a train with the supervisor. Each train has a head and a tail whose positions are defined with references front and rear to class Trackside.

4.2 Translation of the Meta-model

In order to provide the static semantics as B specifications, Meeduse generates machine DSLv0.mch from DSLv0.ecore, the above ecore meta-model. Figure 5 shows the generated Sets, Properties of the Constants, and the related typing invariants. For more details about this translation we refer the reader to [6,7].

4.3 Linkage Machines

Based on the approach of Fig. 2, linkage B machines are created at every iteration. These machines link the existing B specification (*e.g.* machine M0), which we would like to animate, to the DSL components (*e.g.* machine DSLv0). It refines the existing B model and includes the machine issued from the meta-model of the DSL. The idea is to map concepts from the DSL to concepts from the existing B model. In this sub-section, we give an overview of the mappings used in this first refinement level of our case study. More details about the corresponding approach are provided in [15].

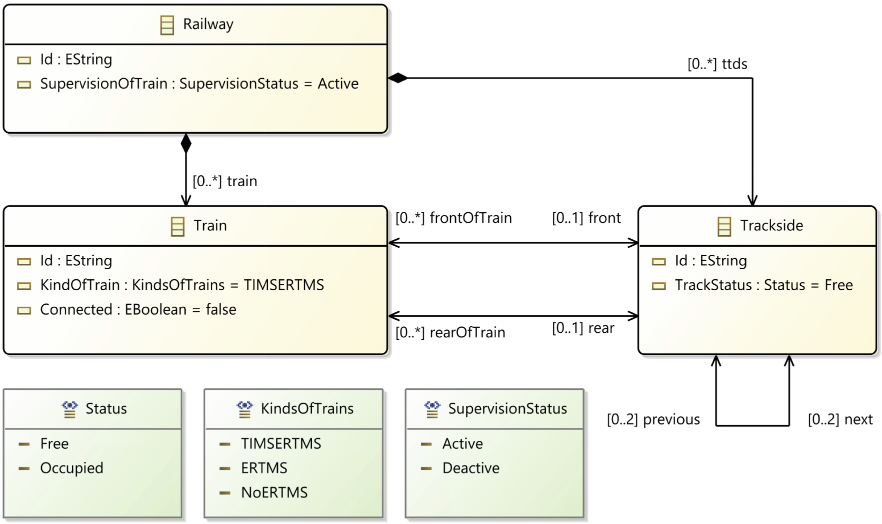


Fig. 4. DSLv0 Meta-Model

Rule 1: EClass to BMachine. In the DSL, class Railway is a root class that contains all other classes. We consider that this class and machine M0 represent the same concept, which is the railway system. Thus, we introduce a constant *Linked_Railway* in the linkage machine which will be helpful later while mapping the underlying concepts.

CONSTANTS: $Linked\_Railway$
PROPERTIES: $Linked\_Railway \in Railway$

Rule 2: Enumeration to BSet. In machine M0, StateTTD is a set containing values freeT and occupiedT. The similar concept in the DSL is the enumeration Status with the values: Free and Occupied. So, the mapping between an enumeration and a set is done as follows:

CONSTANTS: $Linked\_Status$
PROPERTIES: $Linked\_Status{=}\{freeT \mapsto Free,\ occupiedT \mapsto Occupied\}$

Rule 3: EClass to BSet. In the existing model, constant Ttds is set from constant minTTD to constant maxTTD. The Ttds concept is similar to class Trackside from the DSL. Same like Trains is a finite set of TRAINS and the similar concept from the DSL is class Train. Mapping these sets and classes is done as follows:

CONSTANTS: $Linked\_Trackside$, $Linked\_Trains$
PROPERTIES:
$Linked\_Trackside \in Trackside \rightarrowtail\!\!\!\!\twoheadrightarrow Ttds \wedge$
$Linked\_Trains \in Train \rightarrowtail\!\!\!\!\twoheadrightarrow Trains$

Rule 4: Boolean EAttribute to Boolean BVariable . Attribute isConnected of class Train is a boolean attribute. The similar concept in the existing model is variable Connected. To map these two concepts, we introduce the following invariant:

INVARIANT: $Connected = (Linked\_Trains \ ; \ isConnected)$

Rule 5: EAttribute (EnumType) to BVariable (SetValued). stateTTD is a variable in the existing model which is typed as a set-value from set StateTTD. In the DSL, TrackStatus is an enumeration attribute in the class Trackside. We use composition relation (;) in this mapping as:

INVARIANT: $TrackStatus = (Linked\_Trackside \ ; \ stateTTD \ ; \ Linked\_Status)$

Machine
$DSLv0$
SETS
$Status = \{Free, Occupied\}$;
$KindsOfTrains = \{TIMSERTMS, ERTMS, NoERTMS\}$;
$SupervisionStatus = \{Active, Deactive\}$;
$RAILWAY$; $TRACKSIDE$; $TRAIN$;
VARIABLES [. . .]
CONSTANTS [. . .]
PROPERTIES
$Railway \in \mathrm{Pow}\ (RAILWAY) \wedge$
$Trackside \in \mathrm{Pow}\ (TRACKSIDE) \wedge$
$Train \in \mathrm{Pow}\ (TRAIN) \wedge$
$KindOfTrain \in Train \mathrel{+\!\!\!\!\rightarrow} KindsOfTrains \wedge$
$previous\_next \in Trackside \leftrightarrow Trackside \wedge$
INVARIANT
$TrainFront \in Train \mathrel{+\!\!\!\!\rightarrow} Trackside \wedge$
$TrainRear \in Train \mathrel{+\!\!\!\!\rightarrow} Trackside \wedge$
$SupervisionOfTrain \in Railway \mathrel{+\!\!\!\!\rightarrow} SupervisionStatus \wedge$
$TrackStatus \in Trackside \mathrel{+\!\!\!\!\rightarrow} Status \wedge$
$Connected \in Train \mathrel{+\!\!\!\!\rightarrow} \mathbf{BOOL} \wedge$
$\forall\ thePrevious.(\ thePrevious \in \mathbf{ran}(previous\_next)$
$\Rightarrow \mathbf{card}(previous\_next^{\ -1}\ [\{thePrevious\}]) \leq 2) \wedge$
$\forall\ theNext.(\ theNext \in \mathbf{dom}(previous\_next)$
$\Rightarrow \mathbf{card}(previous\_next[\{theNext\}]) \leq 2)$

Fig. 5. Excerpt of the structural part of machine DSLv0

Rule 6: Attribute (EnumType) to a Boolean Variable. The boolean variable supervisor defines the status of the controller (false if not active and true if it is active). In the DSL, the same concept is defined using an attribute SupervisionOfTrain (enumeration SupervisionStatus) in class Railway. Mapping between enumeration values and the boolean values is established by means of the following invariant:

INVARIANT:

$$(\mathit{supervisor} = \mathbf{TRUE} \Rightarrow \mathit{SupervisionOfTrain}(\mathit{Linked\_Railway}) = \mathit{Active})$$
$$\wedge\ (\mathit{supervisor} = \mathbf{FALSE} \Rightarrow \mathit{SupervisionOfTrain}(\mathit{Linked\_Railway}) = \mathit{Deactive})$$

Rule 7: Single Valuated EReference to BVariable. Variable trainOccupationTTDFront of the existing model defines the occupation of train's front on a Ttd. In the DSL, this concept is defined with reference TrainFront from class Train to class Trackside. We introduce the following invariant to map both concepts:

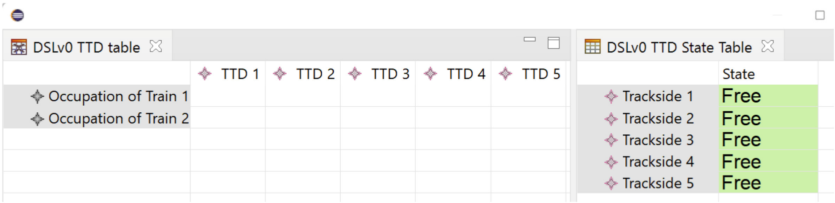
INVARIANT:

$$\mathbf{dom}(\mathit{TrainFront}) = \mathit{Linked\_Trains}^{-1}[\mathbf{dom}(\mathit{trainOccupationTTDFront})]$$

4.4 Modeling and Visual Animation

Figure 6 is a graphical model conforming to DSLv0. It features two TIMS trains (Train 1, Train 2) and five tracks (Trackside 1..5). The state of each track is represented in the right-hand side of the figure. In this model all tracks are set to Free; they are not occupied by any of the two trains. Actually, the model of Fig. 6 is drawn by the domain expert to describe an initial state. Once the linkage machine between DSLv0 and M0 is created, this model can be animated in Meeduse. The tool valuates all the B data (variables and constants), initializes the machine and applies ProB for animation. In a classical animation of B specifications, the B method expert has to complete by hand the specifications with valuations. Here, it is the domain expert who created two instances of Train and five instances of Trackside; and then the tool automatically produces the valuated machine together with its initialisation. Figure 7 shows the movement of trains in the graphical representation by triggering the operations of M0.mch mentioned in Sect. 4.1.

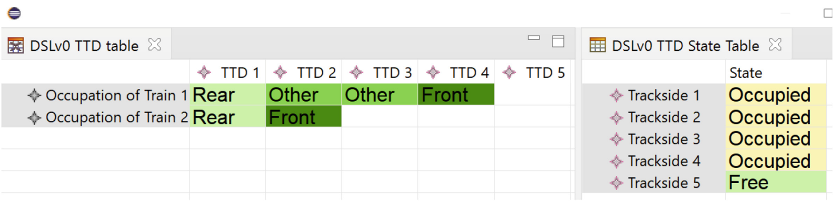
The Front of Train 1 occupies TTD 4 while its Rear occupies TTD 1. The Other shows the occupancy of the train on TTDs that are between Front and Rear. For Train 2, the Front is positioned at TTD 2 and Rear at TTD 1. Apart from Trackside 5, all other tracksides are occupied. At this stage, someone can observe the collision between Train 1 and Train 2 as both occupy the TTD 1 and TTD 2. Actually, the safety property regarding the non-collision is only satisfied in the M3 refinement level of the existing model. We chose not to represent other concepts defined in the meta-model such as supervision status, train connect/disconnect, because they are not directly linked to possible collisions.



	TTD 1	TTD 2	TTD 3	TTD 4	TTD 5
Occupation of Train 1					
Occupation of Train 2					

	State
Trackside 1	Free
Trackside 2	Free
Trackside 3	Free
Trackside 4	Free
Trackside 5	Free

Fig. 6. A model conforming to DSLv0



	TTD 1	TTD 2	TTD 3	TTD 4	TTD 5
Occupation of Train 1	Rear	Other	Other	Front	
Occupation of Train 2	Rear	Front			

	State
Trackside 1	Occupied
Trackside 2	Occupied
Trackside 3	Occupied
Trackside 4	Occupied
Trackside 5	Free

Fig. 7. Animating DSLv0 using M0

5 Application, Findings and Analysis

The complete meta-model of our DSL is shown in Fig. 8. Concepts of every refinement level are presented using different colors. Yellow classes and black associations show DSLv0. Associations in brown color are introduced during DSLv1. Class **VirtualBlock** and associations represented in light blue are from DSLv2. Finally, attributes and associations in purple represent DSLv3.

5.1 Next Iterations

DSLv1. This version is an update of DSLv0 by introducing two new concepts introduced in refinement M1 of the existing specification. Since M1 refines M0, all concepts from M0 are already included in meta-model of DSLv1. Associations frontTrackLocation and rearTrackLocation, illustrated with a brown color in Fig. 8 are added in DSLv1. The frontTrackLocation shows the front location of a train that is communicated to the controller (supervisor) and rearTrackLocation is the communicated location of the train's rear. The inclusion of these concepts updates the graphical representation with Train's location information, as known by the supervisor.

DSLv2. Concepts of DSLv2 are shown in light blue in Fig. 8. Class VirtualBlock has been introduced. It is linked to class Railway using the composition relation. At this level no attributes are included in class VirtualBlock except the Id. A link between class Trackside and VirtualBlock is created where a track side can have many virtual blocks (association virtualblock) and in the opposite each virtual block is associated to at-least one track (association trackside). The other introduced associations in this level are those from class Train to class VirtualBlock

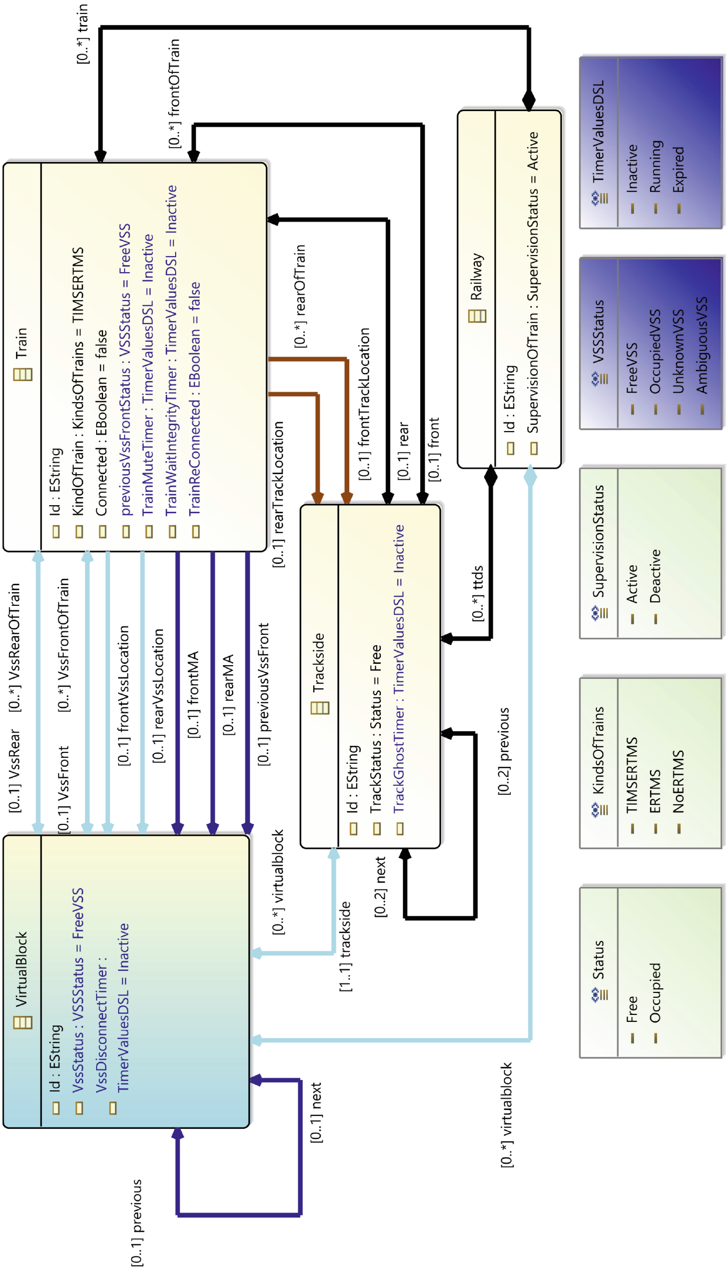
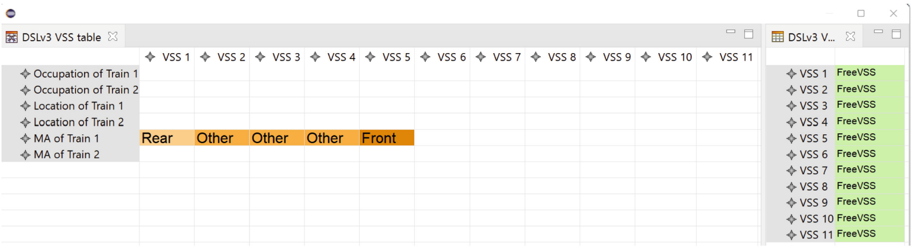


Fig. 8. Whole DSL Meta-Model (Color figure online)

which are VssFront (front of train on virtual block), VssRear (rear of train on virtual block), frontVssLocation (front of train on virtual block communicated to the controller), and rearVssLocation (rear of train on virtual block communicated to the controller). These additional concepts lead to the creation of a new representation containing virtual blocks (VSSs) instead of tracks (TTDs).

DSLv3. Concepts of DSLv3 are shown in violet in Fig. 8. Enumeration VSSStatus gives the state of a virtual block. It can be free (FreeVSS), occupied (OccupiedVSS), unknown (UnknownVSS), or ambiguous (AmbiguousVSS). Enumeration TimerValuesDSL includes values: Inactive, Running and Expired, which are the states of a timer. The associations: previous and next from VirtualBlock to VirtualBlock represents the connections between virtual blocks. The concept of movement authority (MA) is defined by associations: frontMA (MA for train's head) and rearMA (MA for train's rear) from class Train to class VirtualBlock. Association previousVssFront is used to store the value of a previous VSS for a train's front. We introduced the four timer concepts in the DSL as defined in the existing ERTMS/ETCS HL3 specifications. The timers related to the trains are defined in class Train as TrainMuteTimer and TrainWaitIntegrityTimer. The timer related to VSS is VssDisconnectTimer which is introduced inside class VirtualBlock, while timer TrackGhostTimer which is related to TTD is included in class Trackside. The VssStatus attribute in class VirtualBlock shows the status of each virtual block using enumeration VSSStatus. Attribute previousVssFrontStatus stores the status of the previous VSS for the train's front. Finally, attribute TrainReConnected is a Boolean and gets value TRUE when a train is connected back after disconnecting.

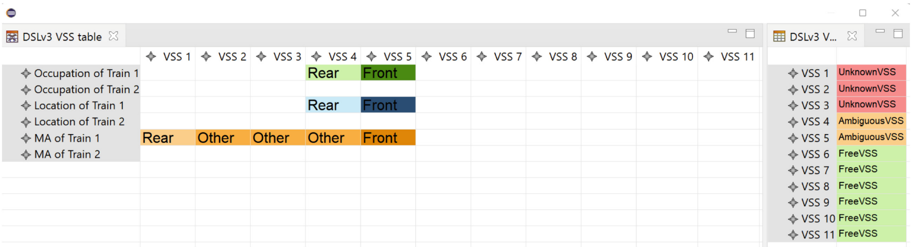
Figure 9 is a graphical representation conforming to DSLv3. The figure represents eleven VSSs and two TIMS trains, in addition to train's location, occupation, and MA. The right-hand side represents the states of the VSSs. Before assigning MA, the train supervisor calculates the VSSs and sets the state of all VSSs to free if there is no train. Once the calculation of VSSs is done, a train can be connected and can be assigned an MA. In Fig. 9 VSSs are free and movements authorities are assigned to Train 1 from VSS 1 till VSS 5. Rear of MA is the "start of authority" and Front of MA is "End of Authority" (EoA).



	VSS 1	VSS 2	VSS 3	VSS 4	VSS 5	VSS 6	VSS 7	VSS 8	VSS 9	VSS 10	VSS 11
Occupation of Train 1											
Occupation of Train 2											
Location of Train 1											
Location of Train 2											
MA of Train 1	Rear	Other	Other	Other	Front						
MA of Train 2											

VSS 1	FreeVSS
VSS 2	FreeVSS
VSS 3	FreeVSS
VSS 4	FreeVSS
VSS 5	FreeVSS
VSS 6	FreeVSS
VSS 7	FreeVSS
VSS 8	FreeVSS
VSS 9	FreeVSS
VSS 10	FreeVSS
VSS 11	FreeVSS

Fig. 9. Assigning MA to Train 1 using DSLv3



	VSS 1	VSS 2	VSS 3	VSS 4	VSS 5	VSS 6	VSS 7	VSS 8	VSS 9	VSS 10	VSS 11
Occupation of Train 1				Rear	Front						
Occupation of Train 2											
Location of Train 1				Rear	Front						
Location of Train 2											
MA of Train 1	Rear	Other	Other	Other	Front						
MA of Train 2											

VSS 1	UnknownVSS
VSS 2	UnknownVSS
VSS 3	UnknownVSS
VSS 4	AmbiguousVSS
VSS 5	AmbiguousVSS
VSS 6	FreeVSS
VSS 7	FreeVSS
VSS 8	FreeVSS
VSS 9	FreeVSS
VSS 10	FreeVSS
VSS 11	FreeVSS

Fig. 10. Train Movement consuming MAs

5.2 Unexpected Behaviors

Figure 10 shows that Train1 has entered and reached EoA. In a normal case, it should be possible to assign new MAs to train 1 to move on the next VSSs, but the animation tells us that this is not possible. The only possible allowed operations are disconnection and connection of Train1. This problem is a deadlock and was identified during the animation of normal scenarios from ERTMS/ETCS. In Fig. 10, a user can observe that some VSSs remain concerned by MAs even after a train has consumed them. It can be seen that VSS 1, 2 and 3 have been released but still they are concerned by the MAs of Train1. This behavior can be considered as a problem from the domain expert point of view. Actually, it reveals some limits (misunderstandings of the requirements) of the existing B specifications.

Another unexpected behavior that we observed is that a VSS never gets the state OccupiedVSS. According to ERTMS/ETCS, when the train's front or rear is over a VSS then the VSS is considered to be occupied. The right side of Fig. 10 shows that VSS 4 and VSS 5 are set to AmbiguousVSS, which should be instead OccupiedVSS as both VSS host the rear and the front of Train1 respectively. In order to test whether this problem is coming from the linkage machine or the existing machine, we analyzed the states of VSSs in the existing machine. We came to the conclusion that the problem is located in the existing machine and this information was communicated to the authors of the existing machine. Note that the authors of the existing model already mentioned in their paper that proof obligations related to VSS state machines were found ambiguous (non-deterministic) and were not discharged.

5.3 Lessons Learned

Lessons Learned from a Formal Methods Expert's Point of View. The existing animation tools (Brama, AnimeB, etc.) do not offer a useful graphical view for the model's animation. Indeed, ProB and AnimB, for instance, only display the values of the variables at each animation step leaving the task of analyzing and explaining them to the user. This is definitely not exploitable for complex systems with several variables like ERTMS 3 system. So, the approach introduced in this paper permits overcoming the drawbacks of the existing tools by providing a useful graphical view of the animation permitting users a better understanding of their systems, and helping them detect errors and bugs. Among others, this approach permits us to detect, for instance, that a VSS never becomes occupied, while this went unnoticed under Rodin and ProB.

Lessons Learned from a Railway Expert's Point of View. Among the three unexpected behaviors, one is to be pointed out at first: VSS never gets the state OccupiedVSS. This is a problem because when a train is on a VSS, this VSS must be occupied. Particularly when the train is connected and sends its position to the control center supervising train movement. In a first analysis, it is surprising that such an evidence is not fulfilled. Analyzing deeper, this is not so surprising. A railway norm is written by railway experts for railway experts. It means that some evidences are not recalled to mind: this is an implicit requirement that everybody is supposed to know.

The paper of Mammar [12], clearly says that in their opinion, the specification is ambiguous. From a methodological point of view, the DSL is run using various scenarios generating behaviors that often surprise a railway expert. Two trains in the same location are correct in the first level because the non-collision mechanisms will be implemented in a later step. The simulation analysis by an expert helps to define the semantic limits of a given DSL but the more interesting part comes when a misunderstanding of the specification is identified. The OccupiedVSS value that is never reached is a good demonstration. Even authors of the code never identified the problem before. It shows that basic specification errors happens and are really difficult to detect without a graphical animation.

5.4 A New Proposed Solution and Proof Obligations

Through an industrial case study (ERTMS 3/ETCS), we showed how a DSL can be used to validate an existing formal B specification. We define links between the concepts defined in our DSLs and those used in the B specification. To verify the various B models, proof obligations (POs) are generated. These POs ensure the correctness of the B specification regarding the linking invariants (see Table 1).

Table 1. Comparison of POs generated using both architectures

Architecture	M0	M1	M2	M3	Linkage0	Linkage1	Linkage2	Linkage3
First Architecture	35	134	153	377	108	168	467	NA
New Architecture	21	83	105	NA	39	120	205	NA

As one can notice, the number of POs is huge; this is due to the fact that the linking invariants of each level include those of the previous one. To overcome this problem, we are working on a new architecture, depicted in Fig. 11), which generates fewer POs and that is even easier to prove.

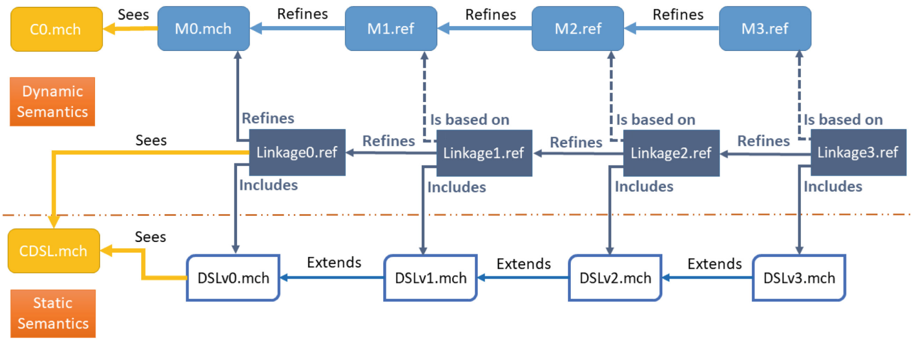


Fig. 11. New Architecture

In this new architecture, first, the concepts of the existing model are kept in a separate context machine called C0.mch and then concepts of the DSL are kept in a context machine called CDSL.mch. In the first iteration, M0.mch sees the C0.mch while DSLv0.mch and Linkage0.mch sees the CDSL.mch. Then the Linkage0.mch refines the M0.mch and includes the DSLv0.mch. In the second iteration, Linkage1.ref (which is based on M1.ref) refines the linkage0.ref and includes the DSLv1.mch (which extends the DSLv0.mch). The same process that is done in the second iteration is applied until the last iteration. The architecture is easier to prove but the animation is complicated using DSLs. In the first architecture shown in Fig. 2, there is one DSL and we are updating the same DSL at each iteration. In the new architecture, at each level, we have a different DSL extending the previous one. The Eclipse Modeling Framework (EMF) [4] does not support such an architecture when designing meta-models of DSLs. But we are able to animate the dynamic semantics of this architecture in Meeduse using a single DSL (using an updated version of DSL at each level). Note that this new architecture is preliminary and not part of this case study.

Currently, the use of this approach on a specification written in a formal language different from the B language incurs additional translation/proof efforts since we have to map the specification into the B language and prove the specification again. For the case study presented in this paper, the B specification provided in [12] has been first rewritten with respect to the B language syntax, and then all the operations have been re-proved under AtelierB by A. Mammar (original author of the existing B specification). Table 1 shows the comparison of number of (POs) generated from both architectures. The table clearly shows that POs generated from the components using the first architecture is significantly higher than the new architecture. AtelierB automatically proved most of the POs. We are not able to generate POs for a few components and in the table, their POs are shown as NA. To reduce this cost, a perspective of this work is to study how the proofs can be performed under Rodin, the Event-B development platform, can be applied to discharge the corresponding proofs under AtelierB.

6 Conclusion

This paper presents an extension of Meeduse, and its application, for validating step-by-step an existing B specification that is developed independently of the Meeduse team. This specification corresponds to the ERTMS/ETCS level 3 case study of the ABZ'2018 conference [2]. The proposed extension brought to the tool the capability to deal with B refinements while animating a DSL. Furthermore, during the application we incrementally built the DSL being guided by the B data that are used in each refinement level. Some specific scenarios were used to show how a visual animation can document border cases of a given refinement level. The railway expert, who is often not an expert in B, is fully aware of the normative framework and may assess and document the hierarchical decomposition with regard to functional safety requirements. An executable DSL allows the railway expert to visualize and discuss requirements such as (non-)collision constraints and train integrity, with the formal methods expert.

While the extension of Meeduse covers B refinements, it does not deal with DSL refinements. This perspective is left to future works. In Sect. 5.4 we discussed a novel architecture and showed that it may reduce the number of POs. However, to make this architecture effective we need to establish a refinement relationship between meta-models of every increment, which requires the extension of the Eclipse Modeling Framework.

References

1. The ERTMS/ETCS signalling system. http://www.railwaysignalling.eu/wp-content/uploads/2016/09/ERTMS_ETCS_signalling_system_revF.pdf. Accessed 16 Mar 2022
2. Butler, M.J., Raschke, A., Hoang, T.S., Reichl, K.: ABZ 2018. LNCS, vol. 10817. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91271-4
3. Collart-Dutilleul, S., Pereira, D.I.A., Bon, P.: Designing operating rules for ERTMS transnational lines. In: Collart-Dutilleul, S. (ed.) Operating Rules and Interoperability in Trans-National High-Speed Rail, pp. 133–161. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-72003-2_6
4. Steinberg, D., Budinsky, F., Marcelo, P., Merks, E.: EMF: Eclipse Modeling Frameworke, 2nd edn. Addison-Wesley, Boston (2009)
5. Ferrari, A., et al.: Survey on formal methods and tools in railways: the ASTRail approach. In: Collart-Dutilleul, S., Lecomte, T., Romanovsky, A. (eds.) RSSRail 2019. LNCS, vol. 11495, pp. 226–241. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-18744-6_15
6. Idani, A.: Meeduse: a tool to build and run proved DSLs. In: Dongol, B., Troubitsyna, E. (eds.) IFM 2020. LNCS, vol. 12546, pp. 349–367. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63461-2_19
7. Idani, A., Ledru, Y., Vega, G.: Alliance of model-driven engineering with a proof-based formal approach. Innov. Syst. Softw. Eng. **16**(3), 289–307 (2020). https://doi.org/10.1007/s11334-020-00366-3
8. Ladenberger, L., Bendisposto, J., Leuschel, M.: Visualising event-B models with B-motion studio. In: Alpuente, M., Cook, B., Joubert, C. (eds.) FMICS 2009. LNCS,

vol. 5825, pp. 202–204. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04570-7_17

9. Ladenberger, L., Leuschel, M.: BMotionWeb: a tool for rapid creation of formal prototypes. In: De Nicola, R., Kühn, E. (eds.) SEFM 2016. LNCS, vol. 9763, pp. 403–417. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41591-8_27
10. Leuschel, M., Butler, M.: ProB: a model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 855–874. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45236-2_46
11. Leuschel, M., Samia, M., Bendisposto, J.: Easy graphical animation and formula visualisation for teaching B (2008)
12. Mammar, A., Frappier, M., Fotso, S.J.T., Laleau, R.: A formal refinement-based analysis of the hybrid ERTMS/ETCS level 3 standard. Int. J. Softw. Tools Technol. Transf. **22**(3), 333–347 (2020). https://doi.org/10.1007/s10009-019-00543-1, https://hal.archives-ouvertes.fr/hal-02975774
13. Servat, T.: BRAMA: a new graphic animation tool for B models. In: Julliand, J., Kouchnarenko, O. (eds.) B 2007. LNCS, vol. 4355, pp. 274–276. Springer, Heidelberg (2006). https://doi.org/10.1007/11955757_28
14. Werth, M., Leuschel, M.: VisB: a lightweight tool to visualize formal models with SVG graphics. In: Raschke, A., Méry, D., Houdek, F. (eds.) ABZ 2020. LNCS, vol. 12071, pp. 260–265. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-48077-6_21
15. Yar, A., Idani, A., Ledru, Y., Collart-Dutilleul, S.: Visual animation of B specifications using executable DSLs. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 617-626. MODELS 2022, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3550356.3561585