Towards a Tool-Based Domain Specific Approach for Railway Systems Modeling and Validation

Akram Idani[1,2(✉)], Yves Ledru[1,2], Abderrahim Ait Wakrime[2], Rahma Ben Ayed[2], and Philippe Bon[2,3]

[1] Univ. Grenoble Alpes, Grenoble INP, CNRS, LIG, 38000 Grenoble, France
{Akram.Idani,Yves.Ledru}@imag.fr
[2] Institut de Recherche Technologique Railenium, 59300 Famars, France
{abderrahim.ait-wakrime,rahma.ben-ayed}@railenium.eu
[3] Univ Lille Nord de France, IFSTTAR, COSYS, ESTAS, 59650 Villeneuve d'Ascq, France
philippe.bon@ifsttar.fr

Abstract. In the railway field, graphical representations of domain concepts are omnipresent thanks to their ability to share standardized information with common knowledge about several railway mechanisms: track circuits, signalling rules... This paper proposes a domain specific approach for railway systems modeling and validation by combining the Model-Driven Engineering (MDE) paradigm and a formal method. First, an example of a graphical DSL is defined thanks to MDE tools, and then the formal B method is used to define its underlying operational semantics and to guarantee the correctness of the model's behaviour with respect to its safety properties. Our approach is assisted by the Meeduse tool which animates and visualizes execution scenarios of domain models. Starting from a given model designed in the DSL tool, Meeduse asks ProB to animate B operations and gets the reached state by means of B variables valuations. Then, it translates back these valuations to the initial DSL resulting in automatic modifications of the domain model. Our approach allows a more pragmatic domain-centric animation than current visual animation techniques since the resulting DSL tool allows domain experts, who are not necessarily trained in formal methods, to design and validate by themselves the various domain models.

Keywords: MDE · DSL · Formal methods · Visual animation

1 Introduction

In railway control and safety systems, the application of formal methods is becoming a strong requirement as recommended by CENELEC EN 50128 standard[1]. However, while formal methods provide solutions to the verification problem, human errors may lead to erroneously validate the specification, and hence

[1] https://standards.globalspec.com/std/13113133/en-50129.

S. Collart-Dutilleul et al. (Eds.): RSSRail 2019, LNCS 11495, pp. 23–40, 2019.
https://doi.org/10.1007/978-3-030-18744-6_2

to produce the wrong system. Indeed, even if formal proofs succeed, a formal specification can be wrong for two main reasons [7]: misunderstandings of the users needs or errors in the expression of these needs. In order to deal with these shortcomings, several formal tools provide graphic animation and visualization techniques [8,14,17] which help in the exploration of alternative behaviors in a step-by-step approach. This technique favours the communication between a formal methods engineer and the domain expert by using domain specific visualizations which is crucial during the validation activity.

Unfortunately, mapping a given graphical representation to the formal specification is a rather time-consuming task (several days or several weeks) and the creation of custom visualizations is often done when the formal model reaches an advanced stage during the modeling process. This is counterproductive since the identification of misunderstandings often leads to enhancements of the formal specifications which in turn impacts the implementation of the visualization. Furthermore, the domain specific visualizations are created by the formal methods engineer who would like to remedy the poor readability of his own specifications and hence, the resulting visualizations may lack of real-user perspective. In [3], Bjørner states that *before we can formulate requirements, we must understand the* [*application*] *domain*, meaning that domain specific representations are required before starting to think about formal models. In a pragmatic approach, these representations should be provided by the domain expert who has a greater knowledge of the application domain than the formal methods engineer.

In the railway domain, specific representations (textual or graphical) of domain concepts are omnipresent thanks to their ability to share standardized information with common knowledge about several railway mechanisms: track circuits, signalling rules, interlocking systems... Nowadays, there are more and more attempts to define DSL tools [12,20,22], based on these specific representations, allowing the domain expert himself to provide useful models to the software system engineer. In this paper, we propose a formal tool-based domain specific approach that defines a DSL for railroad topologies with a concrete graphical syntax and associated formal semantics. The DSL tool is developed in a well-known Model Driven Engineering paradigm (MDE) based on EMF [19] and it is intended to be used by the domain expert in order to design interesting business models. The formal part of our approach is assisted by the Meeduse tool[2] which automatically translates the DSL meta-model into an equivalent B specification [1] gathering the structure of the meta-model as well as basic operations like constructors, destructors, getters and setters. The operational semantics of the DSL are then defined using the formal B method which guarantees the correctness of the model's behavior with respect to its invariant properties. Meeduse allows the animation of underlying execution scenarios using the ProB tool [16]. Starting from a given business model, it asks ProB to animate B operations and retrieves the reached state by means of B variables valuations. Then, Meeduse translates back these valuations to the initial DSL resulting in automatic modifications of

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

the business model which gives rise to a more pragmatic domain-centric animation than current visual animation techniques.

Section 2 provides the static semantics of a railroad DSL done thanks to the MDE paradigm. In Sect. 3, we show how our DSL is enhanced by a formal specification in order to define its operational semantics. Finally Sect. 5 draws the conclusions and the perspectives of this work.

2 A Simple Railroad DSL

The adoption of model-driven engineering (MDE) paradigms in industry is increasing because MDE is assisted by numerous tools for creating and exploiting domain models such as: EMF[3], Xtext[4], Sirius[5], GMF[6], ... These tools had several successful applications thanks to the solutions they provide for rapid-prototyping of DSLs. The application of MDE in order to define DSLs for railway systems promotes readability of these systems and enables stakeholders without experience in programming or formal languages, like certification authorities, to create the models as long as they possess domain knowledge. In MDE, the definition of a DSL follows three steps:

1. The definition of model's semantics via a meta-model which is a central artefact because it allows interoperability between tools such as language analysers (*e.g.* Xtext [2]), code generators (*e.g.* Acceleo [6]), and also model transformation tools (*e.g.* ATL [13]);
2. The expression of contextual constraints using the OCL language in order to enhance the DSL semantics with invariant properties which are not covered structurally by the meta-model;
3. The creation of a palette of concrete syntax elements (textual or graphical) and their relationships with the meta-model.

2.1 Meta-model Definition

Figure 1 gives a simplified meta-model of a DSL dedicated to railroad topologies and signalling systems. The DSL features three main concepts: trains (class Train), sections of a railway track (class Portion), and train movement authority (class MA) which are authorizations given to a train in order to move to a given portion. In this paper, we make some simplifying assumptions such as association between Train and Portion considering that a train occupies a single portion, and then the whole train moves instantly from one portion to another. We also assume that switches move instantly (in practice, this takes about ten seconds). These simplifications do not impact our approach and one could lift them at the price of a more complicated model.

[3] EMF: https://www.eclipse.org/modeling/emf/.
[4] Xtext: https://www.eclipse.org/Xtext/.
[5] https://www.obeo.fr/fr/produits/Eclipse-sirius.
[6] http://www.Eclipse.org/modeling/gmp/.

Classes Light and AutoTrStop define a signalling equipment with traffic lights (in state on or off) and automatic train stop mechanisms which may be armed or disarmed. These devices are associated to the portion where they are located.

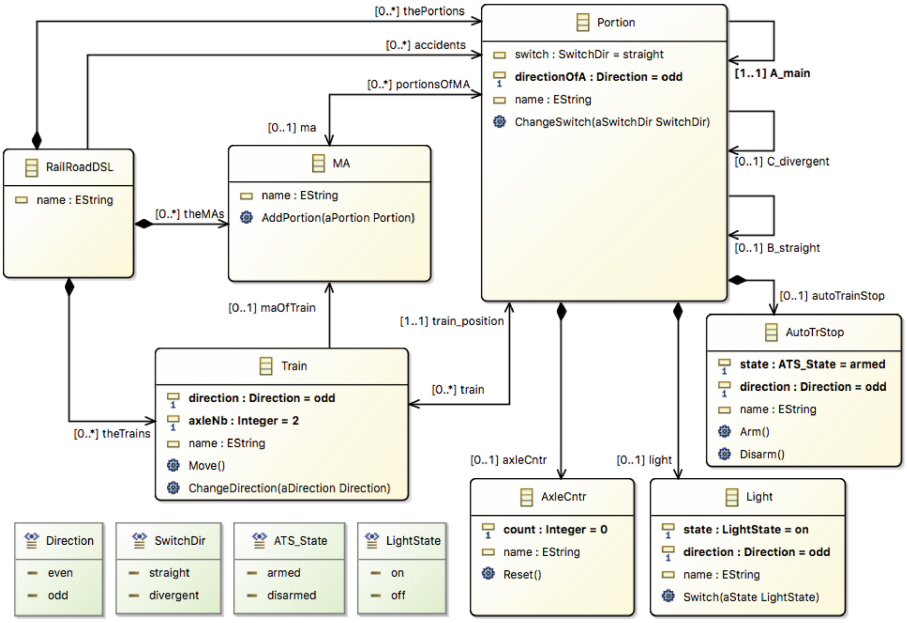

Fig. 1. Simplified meta-model of the RailRoad DSL

2.2 Concrete Syntax Definition

There exists several tools dedicated to the instantiation of meta-models. In this work we used Sirius[7] because its main advantage in comparison with other EMF-based modeling tools is its facility to define conditional styles with an OCL-like syntax. For example, the color representation of a portion would depend on three states like presented in Fig. 2: free, reserved or occupied. States, free and occupied depend on the presence or not of a train over the portion. A portion is called reserved as soon as it is concerned by a movement authority.

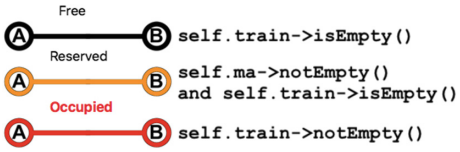

Fig. 2. Portion representations depending on OCL expressions

[7] https://www.obeo.fr/fr/produits/Eclipse-sirius.

The proposed concrete syntax of our Railroad DSL is inspired by graphical representations that we found in several references [9,23,24]. Figure 3 is a snapshot of the resulting DSL tool showing a railroad under construction by a domain expert where five track sections are being assembled.

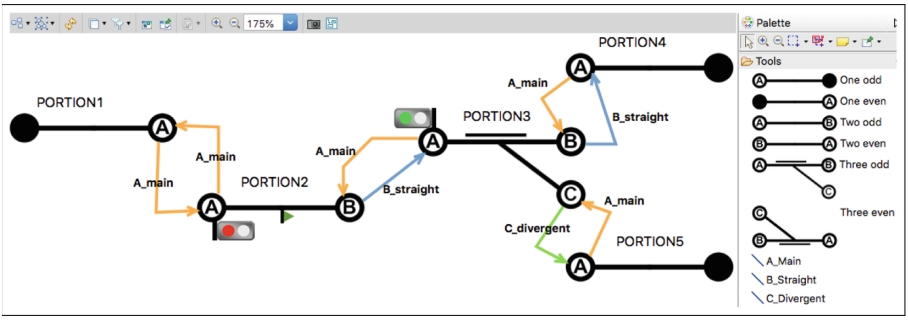

Fig. 3. Railroad under construction in a domain-specific syntax

The meta-model defines three kinds of portions depending on values of associations `A_main`, `B_straight` and `C_divergent`. The first kind of portions represents railroad extremities (*e.g.* portions 1, 4 and 5) and they refer only to their next portion through relation `A_main`. The second kind is a middle horizontal portion like portion 2 that refers to both main and straight portions by means of relations `A_main` and `B_straight`. Finally, the third kind of portions deals with switches such as portion 3 which divides into two: track B (linked to the straight portion) and track C (linked to the diverging portion). The horizontal line represents the state of the switch, pointing towards the straight (B) or diverging (C) end. It depends on values of attribute `switch` in class Portion. In the case of portion 3, this attribute is set to `straight`. Two traffic lights are introduced in this model in order to control the access to portions 2 and 3. Their graphical concrete representations depend on values of attributes `state` (on or off) and `direction` (even or odd) defined in class Light. An automatic train stop (ATS) device is also positioned on portion 2 and it is by default disarmed.

Track layout of Fig. 4 is the final model issued from Fig. 3 after hiding portion connections and where two trains, T1 and T2, are positioned respectively on portions 1 and 5. In this example, the travelling directions are `odd` for T1 and `even` for T2. Note that directions of trains and portions are independent; they don't impact each other. However, the directions of lights are relevant for train movements. Indeed, trains are concerned by lights which are oriented in the same travelling direction.

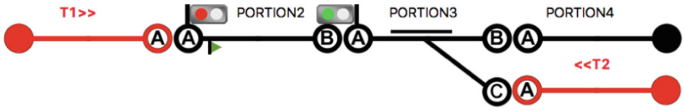

Fig. 4. A simple railroad model with two trains (Color figure online)

2.3 Contextual OCL Constraints

Meta-models are not powerful enough to represent all static semantics of a given DSL. In fact, they define context-free models which are models without any other restrictions than those defined in the meta-model and hence these models are not necessary conformant to the well-definedness rules required by the application context. In our example, the usage context of a railroad requires that tracks follow some rules such as the absence of holes, etc. In order to enhance the static semantics of our DSL, we use the OCL language which is integrated within EMF thanks to the OCLinEcore tool. Our OCL rules deal only with structural aspects of railroads, and basically they define how portions must be linked to each others. Three main invariants are defined depending on portion kinds: `AMainPortion`, `BStraightPortion` and `CDivergentPortion`.

Every portion has a successor portion (called main portion) with respect to association `A_main` in the meta-model of Fig. 1. Invariant `AMainPortion` is the well-definedness rule of this association and assesses that given two portions P1 and P2 such that P2 is the main portion of P1, then if P1 and P2 have opposite directions, then P1 must be also the main portion of P2, otherwise P1 is the straight or the divergent portion of P2. In Fig. 3 for example, PORTION1 and PORTION2 have opposite directions then every portion is the main portion of the other. Based on the same example, PORTION3 is the main portion of portions PORTION4 and PORTION5 and such that the three portions have the same direction, then none of PORTION4 and PORTION5 is the main portion of PORTION3.

```
Context Portion inv AMainPortion:
   (A_main.direction <> {direction} implies A_main.A_main = {self}) and
   (A_main.direction = {direction} implies
               A_main.B_straight = {self} or A_main.C_divergent = {self})
```

Invariants `BStraightPortion` and `CDivergentPortion` define rules for associations `B_straight` and `C_divergent`. They allow to strengthen invariant `AMainPortion` for portions which are not linked via association `A_main`. For example, if two portions P1 and P2 have opposite directions and such that P2 is the straight or the divergent portion of P1 then P1 must be either the straight or the divergent portion of P2.

```
Context Portion inv BStraightPortion:
self.B_straight -> notEmpty() implies
```

```
    (B_straight.direction <> {direction} implies
         B_straight.B_straight = {self} or B_straight.C_divergent = {self})
    and (B_straight.direction = {direction} implies A_main.A_main = {self})
Context Portion inv CDivergentPortion:
self.C_divergent -> notEmpty() implies
    (C_divergent.direction <> {direction} implies
         C_divergent.B_straight = {self} or C_divergent.C_divergent = {self})
    and (C_divergent.direction = {direction} implies A_main.A_main = {self})
```

The EMF platform provides a validation mechanism that checks OCL invariants provided a given input model. Figure 4 is a valid model with respect to the above invariants. This validation is interesting for the domain expert who defines informally the various management rules and who becomes able thanks to the tool to check their validity on his own models. Note that railway domain experts are not intended to write OCL expressions by themselves. In fact, the DSL tool development is the task of MDE experts who has the ability to define metamodels with associated static constraints.

2.4 Discussion

This section has shown how the MDE paradigm with associated tools is applied in order to develop a DSL for the railway domain. At this stage we didn't yet start the creation of a formal model contrary to classical techniques where the development process starts by a formal language. Our approach starts by the definition of a DSL tool like that presented in this section which allows to efficiently involve the domain expert in the development process.

Thanks to the DSL tool, the domain expert becomes able to provide various domain representations (*e.g.* Figs. 4 and 5) and also to check whether the associated contextual rules are respected or not. For example, based on the model of Fig. 4, the domain expert can informally explain that if a train T1 is located on PORTION1, it cannot move to PORTION2 due to the red light. This would allow an other train T2 located on PORTION5 to go to PORTION4 after crossing first PORTION3 and next PORTION2 where it will be able to change its direction. The straight direction of the switch allows T2 to reach PORTION4 when it comes from PORTION2. If train T1 violates the red signal the ATS should be armed automatically and the train will be stopped over PORTION2.

In general, movements of trains are more difficult to represent than this simple informal description, because in addition to the complexity of realistic railway track layouts (like that of Fig. 5) and the corresponding signalling systems, they also refer to movement authority given by traffic agents to the train drivers. In order to play useful scenarios from a domain-centric point of view, the DSL must be enhanced by behavioural aspects showing how routes are assigned to trains and how these trains can move in a safe (or unsafe) way.

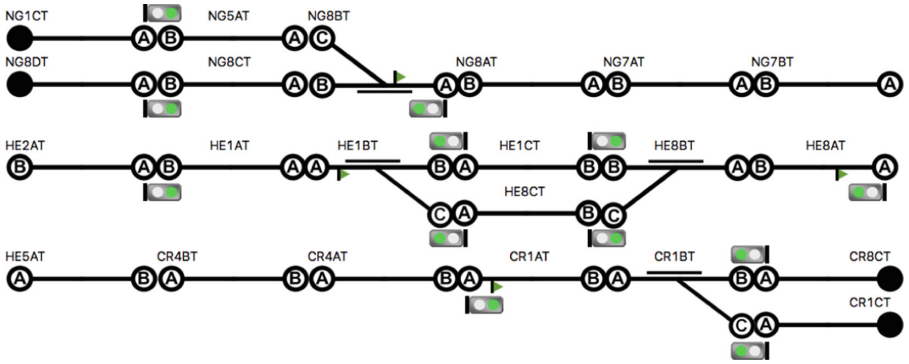

Fig. 5. Realistic example inspired by [24] (Color figure online)

3 Formal Operational Semantics

Operational semantics of our DSL are structured into several formal models which are linked using the inclusion mechanism of the B method: (*i*) a functional model which is automatically extracted from the meta-model, (*ii*) a safety-free model in which train accidents may happen, and (*iii*) a safe model applying authorization rules in order to control the train movements and avoid critical situations.

Our approach, summarized by Fig. 6, first translates the meta-model into a functional formal B specification using a UML-to-B transformation technique [10]. Then starting from a given instance of this meta-model, we apply the Meeduse tool in order to animate domain-centric scenarios based on the operational semantics defined in the formal models. The tool injects any valid instance of a meta-model into the functional specification by applying valuations to its variables. In Meeduse, animation of B specifications is done using the ProB tool [16]. Meeduse asks ProB to animate B operations and gets the new variable valuations and then it translates back these valuations to the initial graphical model

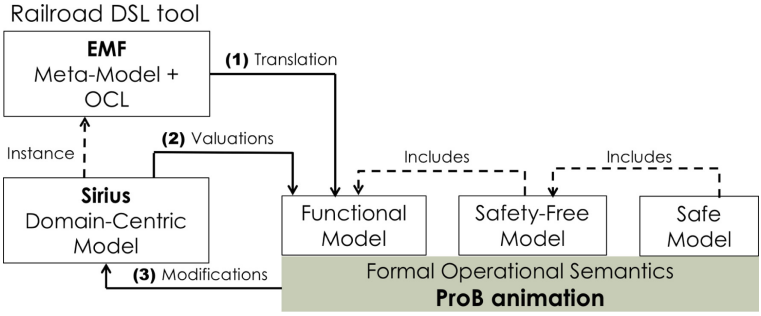

Fig. 6. Overall methodology

resulting in an automatic visual animation. Demonstration videos of Meeduse with graphical and textual DSL animation can be found at: http://vasco.imag.fr/tools/meeduse/.

3.1 Functional Formal Model

In this step we use the B4MSecure platform [10] which translates the structural aspects of the meta-model as follows:

- A meta-class *Class* gives an abstract set named $CLASS$ representing possible instances and a variable named *Class* representing the set of existing instances such that existing instances belong to the set of possible instances.
- An enumeration is translated into a enumerated set (*e.g.* LightState).
- Basic types (*e.g.* integer, boolean) become B types (Z, Bool, ...).
- Attributes and references lead to functional relations.

Figure 7 shows the declarative part of the B specification extracted from classes Train, Portion as well as the association between these classes. In this specification single valuated features are represented by partial or total functions with respect to their optional/mandatory character. For example, the `A_main` feature of a portion is single-valuated and mandatory (it leads to a total function) contrary to features `B_straight` and `C_divergent` which are optional (they lead to partial functions).

MACHINE
Functional
SETS
$PORTION$; $TRAIN$;
$SwitchDir = \{straight, divergent\}$;
$Direction = \{even, odd\}$;
VARIABLES
Portion, *Train*,
TrainOfPortion,
Portion_switch,
Portion_directionOfA,
Portion_A_main,
Portion_B_straight,
Portion_C_divergent,
Train_direction

INVARIANT
$Portion \in \mathcal{F}\ (PORTION) \wedge$
$Train \in \mathcal{F}\ (TRAIN) \wedge$
$TrainOfPortion \in Train \rightarrow Portion \wedge$
$Portion\_switch \in Portion \nrightarrow SwitchDir \wedge$
$Portion\_directionOfA \in Portion \rightarrow Direction \wedge$
$Portion\_A\_main \in Portion \rightarrow Portion \wedge$
$Portion\_B\_straight \in Portion \nrightarrow Portion \wedge$
$Portion\_C\_divergent \in Portion \nrightarrow Portion \wedge$
$Train\_direction \in Train \rightarrow Direction$

Fig. 7. Subset of the generated functional machine

The behavioural part of the functional B machine provides all basic operations such as getters, setters, constructors and destructors. For example, operation `Train_SetTrain_position` of Fig. 8 is the basic setter of feature `train_position` associated to class Train in the meta-model. This operation

Train_SetTrain_position(*aTrain*,*aTrain_position*) =
PRE
aTrain ∈ *Train* ∧
aTrain_position ∈ *Portion* ∧
{(*aTrain* ↦ *aTrain_position*)} ⊈ *TrainOfPortion*
THEN
TrainOfPortion := ({*aTrain*} ⩤ *TrainOfPortion*)
∪ {(*aTrain* ↦ *aTrain_position*)}
END

Fig. 8. B Setter of feature `train_position` in class Train

puts a train (parameter `aTrain`) on any portion (parameter `aTrain_position`) provided that the portion is different from the current train position.

From our meta-model, B4MSecure produced a B specification whose length is about 900 lines with 29 variables, 73 operations for which the Atelier B prover generated 127 proof obligations that it was able to prove automatically. In fact, operations produced by B4MSecure are correct by construction with respect to the typing invariants generated automatically from the meta-model structure. The introduction of additional invariants requires improvements of operations that may violate them. There are two kinds of invariants: those about the railroad topology, and those that deal with train movements. In this work it is not necessary to use B in order to specify the well-definedness rules like those expressed in OCL. Indeed, our formal operational semantics focus on train behaviours which are operations that don't modify the railroad topology. As domain models are provided by the domain expert and validated thanks to the EMF validation mechanism based on OCL constraints, we have the guarantee that trains would not move over deficient railroads and hence we choose to keep this functional B machine as simple as possible. Train behaviours will be specified together with their corresponding invariants in the two other formal specifications defining the DSL operational semantics. The functional machine provides, on the one hand, data structures which conform to the meta-model and, on the other hand, utility operations useful for the definition of train routes and movements. Portions where accidents happen are defined by reference `accidents` in the meta-model and the corresponding B structure is variable `Portion_accidents` defined as: *Portion_accidents* ⊆ *Portion*. Operation `Portion_SetAccidents` is its basic update operation (Fig. 9):

Portion_SetAccidents(*theAccidents*) =
PRE *theAccidents* ∈ $\mathcal{F}$ (*Portion*)
THEN *Portion_Accidents* := *theAccidents*
END

Fig. 9. B Setter for accidents

3.2 Safety-Free Formal Model

In general, the safety of a railway system is defined by a set of operating rules that must be followed by railway agents, like stopping the train when the light is red. Unfortunately several real situations show that human errors (accidental or intentional) can lead to rule violations and hence to accidents. The safety-free operational semantics address behaviours which are uniquely governed by the laws of physics. For example, if physical devices, like ATS in our DSL, are not actioned in order to block a train, then the train has the ability to move and may induce accidents. We define the following B operations:

- `Portion_ChangeSwitch`: given a portion *aPortion* with a switch like PORTION3 in Fig. 4 (*i.e.* $aPortion \in dom(Portion\_C\_Divergent)$), this operation changes the switch direction (straight or divergent), or leads to an accident if the portion is occupied ($TrainOfPortion^{-1}[\{aPortion\}] \neq \emptyset$). Figure 10 shows the effect of this operation on PORTION3 starting from two different initial states: free and occupied by train T1.
- `Train_ChangeDirection`: changes the direction of a train *aTrain* from even to odd and vice-versa, or produces an accident if the train is located on a switch portion ($TrainOfPortion(aTrain) \in dom(Portion\_C\_Divergent)$). Our DSL semantics assume that it is dangerous to change the direction of a train on a switch. A safe scenario is when the train leaves the switch portion before it changes its direction, otherwise an accident happens.
- `MA_AddPortion`: adds a movement authority *aMA* to a portion *aPortion* provided that the portion is not already concerned by a movement authority ($PortionMA[\{aPortion\}] = \emptyset$)[8]. Considering that *aMA* is linked to one train, this operation is useful in order to create train routes.
- `Light_Switch`: switches a light from red to green and vice-versa.
- `AutoTrStop_Arm`: arms and disarms an ATS.
- `Train_Move`: moves a train from a portion P1 to a portion P2 provided that P1 is not concerned by an accident and also the ATS (if it exists) of P2 is disarmed. This operation may produce accidents in two cases: derailment if the train tries to leave a track extremity in the wrong direction, or collision if the train enters on a portion occupied by an other train.

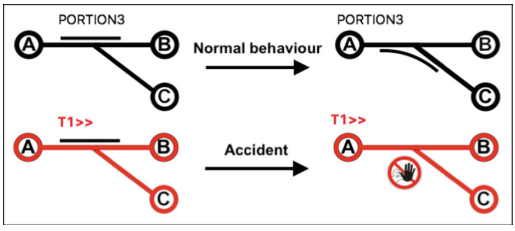

Fig. 10. Animation of operation *Portion_ChangeSwitch*(*PORTION3*) (Color figure online)

[8] *PortionMA* is a partial function mapped from the association between classes MA and Portion.

For space reasons, in the following we'll focus only on the specification of operation `Train_Move`. Given a train *aTrain* ($aTrain \in Train$), this operation is feasible under two preconditions:

Precondition (`Pre`$_1$): the current portion is not concerned by an accident, *i.e.* it does not belong to set *Portion_Accidents*.

$$TrainOfPortion(aTrain) \notin Portion\_Accidents$$

Precondition (`Pre`$_2$): if the current portion is associated to an ATS then the ATS has a different direction than the train or it is disarmed.

$$\begin{array}{l}((TrainOfPortion(aTrain) \in \mathbf{ran}(ATSOfPortion)) \Rightarrow (\\ \quad (AutoTrStop\_direction(ATSOfPortion^{-1}(TrainOfPortion(aTrain))) \\ \qquad\qquad \neq Train\_direction(aTrain)) \\ \quad \vee \quad (AutoTrStop\_state(ATSOfPortion^{-1}(TrainOfPortion(aTrain)))=disarmed)) \\ )\end{array}$$

Actions of operation `Train_Move` address several situations depending on the current portion of the train and also the portion to which the train is intended to move. First we compute, based on two definitions *next_portion_odd* and *next_portion_even*, the next portion with respect to the traveling direction of a train (even or odd) and also to the portion connexions whose semantics were defined by the OCL invariants. In the following we present only *next_portion_odd* since *next_portion_even* is analog.

$$\begin{array}{l}next\_portion\_odd == \{p1, p2 \mid p1 \in Portion \wedge p2 \in Portion \\ \quad \wedge (p1 \notin \mathbf{dom}(Portion\_B\_straight) \\ \qquad \Rightarrow (Portion\_directionOfA(p1) = even \wedge p2 = Portion\_A\_main(p1))) \\ \quad \wedge ((p1 \in \mathbf{dom}(Portion\_B\_straight) \wedge p1 \notin \mathbf{dom}(Portion\_C\_divergent)) \\ \qquad \Rightarrow ((Portion\_directionOfA(p1) = odd \wedge p2 = Portion\_B\_straight(p1)) \\ \qquad\quad \vee (Portion\_directionOfA(p1) = even \wedge p2 = Portion\_A\_main(p1)))) \\ \quad \wedge (p1 \in \mathbf{dom}(Portion\_C\_divergent) \\ \qquad \Rightarrow ((Portion\_directionOfA(p1) = odd \\ \qquad\quad \wedge ((Portion\_switch(p1)=straight \wedge p2 = Portion\_B\_straight(p1)) \\ \qquad\qquad \vee (Portion\_switch(p1)=divergent \wedge p2 = Portion\_C\_divergent(p1))) \\ \qquad\qquad \vee (Portion\_directionOfA(p1) = even \wedge p2 = Portion\_A\_main(p1))))) \\ \}\end{array}$$

Given the railroad of Fig. 4, the application of relations *next_portion_odd* and *next_portion_even* to PORTION3 gives respectively PORTION4 and PORTION2. Relation giving all next portions is thus a partial function defined as:

$$\begin{array}{l}next\_portion == \{dd, np \mid dd \in Direction \wedge np \in Portion \nrightarrow Portion \\ \qquad\qquad\qquad\qquad \wedge (dd = even \Rightarrow np = next\_portion\_even) \\ \qquad\qquad\qquad\qquad \wedge (dd = odd \Rightarrow np = next\_portion\_odd)\ \}\end{array}$$

Definition (`Def`$_1$): *curr_portion* is the portion on which *aTrain* is positioned:

$$curr\_portion = TrainOfPortion(aTrain)$$

Definition (`Def`$_2$): *nxt_port* is the portion to which *aTrain* should move:

$$nxt\_port = (next\_portion(Train\_direction(aTrain)))(curr\_portion)$$

Condition (`Accident`$_1$): defines a situation where the train derails.

$$(Train\_direction(aTrain) = even \wedge curr\_portion \notin \mathbf{dom}(next\_portion\_even)) \\ \vee (Train\_direction(aTrain) = odd \wedge curr\_portion \notin \mathbf{dom}(next\_portion\_odd))$$

Condition (`Accident`$_2$): the next portion is already occupied by an other train.

$$\mathbf{card}(TrainOfPortion^{-1}[\{nxt\_port\}]) > 0$$

Condition (`MoveAuthorization`): the train enters into a portion on which it has a movement authority.

$$aTrain \in \mathbf{ran}(TrainMA) \wedge nxt\_port \in \mathbf{dom}(PortionMA) \wedge \\ PortionMA(nxt\_port) = TrainMA^{-1}(aTrain)$$

Operation `Train_Move`, presented below, moves the train from one portion to an other, by applying operation `Train_SetTrain_position`, or produces an accident using operation `Portion_SetAccidents`. If the train enters a portion for which it has a movement authority then the authority is consumed using operation `MA_RemovePortionsOfMA` which removes a link between a portion and a movement authority. These operations are a part of the basic operations provided by the functional B specification.

```
Train_Move(aTrain) ==
PRE
    aTrain ∈ Train ∧ (Pre1) ∧ (Pre2)
THEN
    LET curr_portion BE (Def1) IN
        IF (Accident1) THEN
            Portion_SetAccidents(Portion_Accidents ∪ {curr_portion})
        ELSE
            LET nxt_port BE (Def2) IN
                Train_SetTrain_position(aTrain, nxt_port);
                IF (Accident2) THEN
                    Portion_SetAccidents(Portion_Accidents ∪ {nxt_port})
                END ;
                IF (MoveAuthorization) THEN
                  MA_RemovePortionsOfMA(PortionMA(nxt_port), nxt_port)
                END
            END
        END
    END
END
```

3.3 Safe Formal Model

Operational semantics of the safety-free model allow the domain expert to visualize critical situations and simulate the corresponding scenarios in the DSL tool. In fact, in the safety-free model the driver is able to override movement authority and traffic lights. For example, given the model of Fig. 4 animation of operation `Train_Move(T1)`, moves train T1 from PORTION1 to PORTION2 which means that the driver violated two safety rules: the red light of PORTION2 and the entry into a portion without an authorization. Furthermore, as the ATS of PORTION2 is disarmed, the train can continue its way.

Operational semantics defined by the safe formal model apply restrictions to operations of the safety-free model in order to keep behaviours without accidents and take into account authorizations given by the railway operating rules. First, conditions $\texttt{(Accident}_1\texttt{)}$ and $\texttt{(Accident}_2\texttt{)}$ must be false, and then condition `(MoveAuthorization)` must be true, meaning that the train cannot move if the next portion is not concerned by any movement authority or the movement authority associated to the next portion concerns an other train. In addition to the portion reservation mechanism assured by movement authority the driver must also respect signalling rules.

Condition `(LightAuthorization)`: means that if the portion to which the train should move is concerned by a light, then this light is either oriented to the opposite direction than that of the train or it is green.

$$nxt\_port \notin \mathbf{ran}(LightOfPortion) \lor Light\_direction(LightOfPortion^{-1}(nxt\_port)) \neq Train\_direction(aTrain) \lor Light\_state(LightOfPortion^{-1}(nxt\_port)) = on$$

The safe version of `Train_Move`, named `Safe_Train_Move`, restricts the call to `Train_Move` by grouping all safety conditions in its precondition:

Safe_Train_Move(*aTrain*) ==
PRE
  aTrain ∈ *Train* ∧ $\texttt{(Pre}_1\texttt{)}$ ∧ $\texttt{(Pre}_2\texttt{)}$ ∧
  LET *curr_portion* **BE** $\texttt{(Def}_1\texttt{)}$ **IN**
    not$\texttt{(Accident}_1\texttt{)}$ ∧
    LET *nxt_port* **BE** $\texttt{(Def}_2\texttt{)}$ **IN**
      not$\texttt{(Accident}_2\texttt{)}$ ∧ (MoveAuthorization) ∧ (LightAuthorization)
    END
  END
THEN
  Train_Move(*aTrain*)
END

The safe formal model applies the same principles than those discussed for operation `Train_Move`, to the other operations and also introduces safety invariants such as $Portion\_Accidents = \emptyset$, which guarantees the absence of accidents. Animation of the safe operations in Meeduse gives the possibility for the domain

expert to attest whether the railway operating rules as specified in B, are valid or not. A by-product of validation through simulation is that it allows also to detect availability bugs. Indeed, it is quite easy to build a safe system, just prevent the trains and switches from moving. The use of simulation allows the domain experts to also assess the availability of the safe system. Note that validation by proofs and model-checking of the safe model is discussed in [15].

Figure 11 gives different states of the domain model (left hand side) with the list of B operations (right hand side) that can be enabled by the animator at every state. In the first state, on top of this figure, the domain expert can: (*i*) change the direction of trains TRAIN1 and TRAIN2; (*ii*) change the switch from straight to divergent; (*iii*) arm the ATS of PORTION2; and (*iv*) compose train routes using instances of operation `Safe_MA_AddPortion`. In this situation the light cannot be turned to green, it can only be kept to red due to the rules that we considered for this example. Animation of operation `Safe_MA_AddPortion(MA2,PORTION3)` followed by `Safe_MA_AddPortion(MA2,PORTION2)`, reaches the state presented in the middle of Fig. 11 where the color of PORTION2 and PORTION3 became orange meaning that these portions are reserved for some train. In this state, operation `safe_Train_Move` can be enabled in order to start moving TRAIN2. Since PORTION2 and PORTION3 are reserved for TRAIN2, operation `Safe_Train_Move(TRAIN2)` can be animated twice which leads to the state in bottom of Fig. 11 where TRAIN2 occupies PORTION2 after consuming the authorizations provided by its route.

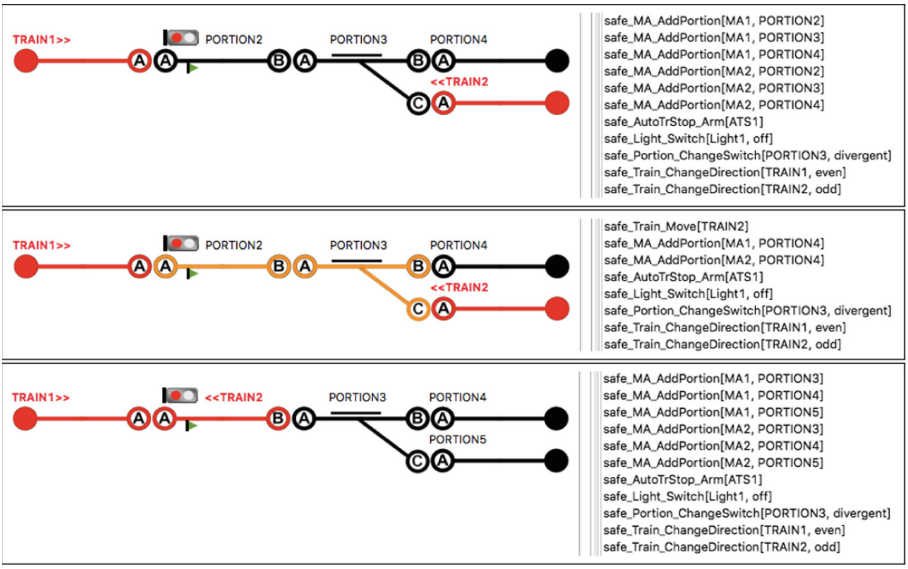

Fig. 11. Meeduse animation of a railroad model with safe operational semantics (Color figure online)

4 Related Works

This paper has shown the application of our approach and its tool support, to the railway field using a simple railroad DSL. In a more general context, besides the contributions discussed along the paper to visual animation techniques [8,14,17], our work presents an advancement in comparison with existing approaches [4,21] where DSLs are mixed with formal methods. In fact, in these works, once the formal model is defined (manually [4] or semi-automatically [21]), they don't offer any way to animate jointly the formal model and the domain model. Often translation techniques, start from a DSL definition and then they get lost in the formal process. In [21], the authors propose to use classical visual animation by applying BMotion Studio [14] to the formal specifications. Unfortunately, this is not only time consuming but also requires some additional verifications in order to address the compatibility between the initial DSL and the graphical representations used in BMotion Studio. Our approach applies well-known MDE tools for DSL creation (EMF, OCLinEcore and Sirius) and automatically manages the traceability between the formal model and the domain model.

The extraction of B specifications from a meta-model applies a UML-to-B translation technique using the B4MSecure tool [10]. The advantage of B4MSecure in comparison with other UML-to-B tools [5,18] is that it offers an extensibility facility allowing to easily add new UML-to-B rules or to modify existing rules depending on the application context. B4MSecure is an open-source MDE platform which gathers several transformations and hence it allows to experiment transformations issued from various approaches. An other technical advantage of B4MSecure with respect to other tools is that it generates a trace file in which links between the initial UML model and the resulting B specification are registered. In order to translate back a state computed by ProB to the initial DSL model, Meeduse requires such a trace file with a corresponding meta-model. Finally, Meeduse is conceived for formal (graphical or textual) DSL definition, while existing tools [5,18] are concerned by UML notations only. We have experimented Meeduse on several DSLs: petri-nets, light regulator, process scheduler, tic-tac-toe, puzzle game, lift, family model... The results were concluding for a rigorous MDE development with end-user validation. It was also interesting for debugging the formal specifications thanks to the joint domain views.

5 Conclusion

In the railway field, there are more and more attempts to define domain specific models based on graphical representations [12,20,22]. For example the RailTopoModel initiative introduced in 2013 [9] gives a common visual standard in railway infrastructure modelling. Unfortunately, these DSLs lack of formal operational semantics and hence they don't apply reasoning tools to address the correctness of their dynamic aspects. Our technique addresses this challenge and allows domain experts, without any knowledge in formal methods, to design

railway models in a formally defined DSL and then to simulate safety-critical behaviours like those producing accidents. The SafeCap platform [11] proposes a railway DSL with formal static semantics using the EMF framework. Since Meeduse fits well to EMF-based DSLs, we think that it can contribute to this platform in order to formally specify and simulate its operational semantics.

This mix of MDE and B has several perspectives. In addition to the application to existing railway standards like ERTMS/ETCS, we plan to address multi-views modeling and interactions between various models. Indeed, a railway DSL can be better structured into several views which can be animated together in Meeduse: driver views showing train interactions with signalling systems, traffic agent views managing movement authorizations and train routes, global views...

Acknowledgments. This work is funded by the NExTRegio project of IRT Railenium. The authors would like to thank SNCF Réseau for its support. We also thank German Vega for his contributions to B4MSecure and Meeduse.

References

1. Abrial, J.-R.: The B-book: Assigning Programs to Meanings. Cambridge University Press, New York (1996)
2. Bettini, L.: Implementing Domain-Specific Languages with Xtext and Xtend. Packt Publishing, Birmingham (2013)
3. Bjørner, D.: Rôle of domain engineering in software development—why current requirements engineering is flawed !. In: Pnueli, A., Virbitskaite, I., Voronkov, A. (eds.) PSI 2009. LNCS, vol. 5947, pp. 2–34. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11486-1_2
4. Bodeveix, J.-P., Filali, M., Lawall, J., Muller, G.: Formal methods meet domain specific languages. In: Romijn, J., Smith, G., van de Pol, J. (eds.) IFM 2005. LNCS, vol. 3771, pp. 187–206. Springer, Heidelberg (2005). https://doi.org/10.1007/11589976_12
5. Dghaym, D., Poppleton, M., Snook, C.: Diagram-led formal modelling using iUML-B for hybrid ERTMS level 3. In: Butler, M., Raschke, A., Hoang, T.S., Reichl, K. (eds.) ABZ 2018. LNCS, vol. 10817, pp. 338–352. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91271-4_23
6. Eclipse. Acceleo (2012). http://www.eclipse.org/acceleo/
7. Gaudel, M.C.: Advantages and limits of formal approaches for ultra-high dependability. Predictably Dependable Computing Systems. ESPRIT BASIC, pp. 241–251. Springer, Berlin (1995)
8. Hallerstede, S., Leuschel, M., Plagge, D.: Validation of formal models by refinement animation. Sci. Comput. Program. **78**(3), 272–292 (2013)
9. Hlubuek, A.: RailTopoModel and RailML 3 in overall context. Acta Polytech. CTU Proc. **11**, 16 (2017)
10. Idani, A., Ledru, Y.: B for modeling secure information systems. In: Butler, M., Conchon, S., Zaïdi, F. (eds.) ICFEM 2015. LNCS, vol. 9407, pp. 312–318. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25423-4_20

11. Iliasov, A., Lopatkin, I., Romanovsky, A.: The SafeCap platform for modelling railway safety and capacity. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) SAFECOMP 2013. LNCS, vol. 8153, pp. 130–137. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40793-2_12
12. James, P., Knapp, A., Mossakowski, T., Roggenbach, M.: Designing domain specific languages – a craftsman's approach for the railway domain using CASL. In: Martí-Oliet, N., Palomino, M. (eds.) WADT 2012. LNCS, vol. 7841, pp. 178–194. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37635-1_11
13. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: ATL: A QVT-like transformation language. In: 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications, OOPSLA 2006, USA, pp. 719–720. ACM (2006)
14. 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
15. Ledru, Y., Idani, A., Ben-Ayed, R., Ait Wakrime, A., Bon, P.: A separation of concerns approach for the verified modelling of railway signalling rules. In: International Conference on Reliability, Safety, and Security of Railway Systems - RssRail 2019, Lille, France, June 2019
16. Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. STTT **10**(2), 185–203 (2008)
17. Li, M., Liu, S.: Integrating animation-based inspection into formal design specification construction for reliable software systems. IEEE Trans. Reliab. **65**, 1–19 (2015)
18. Snook, C., Savicks, V., Butler, M.: Verification of UML models by translation to UML-B. In: Aichernig, B.K., de Boer, F.S., Bonsangue, M.M. (eds.) FMCO 2010. LNCS, vol. 6957, pp. 251–266. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25271-6_13
19. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0, 2nd edn. Addison-Wesley, Reading (2009)
20. Svendsen, A., Haugen, Ø., Møller-Pedersen, B.: Synthesizing software models: generating train station models automatically. In: Ober, I., Ober, I. (eds.) SDL 2011. LNCS, vol. 7083, pp. 38–53. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25264-8_5
21. Tikhonova, U., Manders, M., van den Brand, M., Andova, S., Verhoeff, T.: Applying model transformation and Event-B for specifying an industrial DSL. In: MoDeVVa@ MoDELS, pp. 41–50 (2013)
22. Vu, L.H., Haxthausen, A., Peleska, J.: A domain-specific language for railway interlocking systems. In: 10th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems, pp. 200–209, January 2014
23. Wikipedia. Railroad switch (2015). https://en.wikipedia.org/wiki/Railroad_switch
24. Winter, K., Robinson, N.J.: Modelling large railway interlockings and model checking small ones. In: ACSC, Adelaide, South Australia, February 2003, volume 16 of CRPIT, pp. 309–316. Australian Computer Society (2003)