Incremental Development of a Safety Critical System Combining formal Methods and DSMLs
– Application to a Railway System –

Akram Idani[1,2(✉)], Yves Ledru[1,2], Abderrahim Ait Wakrime[2], Rahma Ben Ayed[2], and Simon Collart-Dutilleul[2,3]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, 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, 59666 Villeneuve d'Ascq Cedex, France
simon.collart-dutilleul@ifsttar.fr

Abstract. In order to assist domain experts, several tools exist for the definition of graphical or textual domain specific modeling languages (DSMLs). The resulting models are useful, but not sufficient, for an overall understanding of the system, especially when formal methods are being applied. Indeed, formal methods failures often result from misunderstandings of the requirements, even if the system is entirely proved. This is confirmed by several industrial experiments which showed that the poor readability of the formal notations is not convenient for communication with domain experts and hence the validation activity is often tedious, time consuming and complex. In order to circumvent this shortcoming, we propose to make domain specific models provable and also executable thanks to the animation of their expected behaviour directly in a dedicated DSML tool. Our approach starts from an intuitive description of the system's operational semantics thanks to high-level Petri-nets which abstract away structural constraints and focus on safety-critical behaviours. Then we take benefit of the B method in order to refine and prove these operational semantics on the one hand, and to merge them with the static semantics of a given DSML, on the other hand. This work is applied to the design of ERTMS/ETCS 3 which is an emergent solution for railway system management.

1 Introduction

Application of formal methods in industrial critical systems became a strong requirement due to their ability to guarantee a zero-fault development. Many well-known success stories can be cited especially in the railway domain [13], like for example Meteor, the automated Paris subway. However, formal methods also suffer from the poor readability of their notations [6] which is not convenient for validation. In fact, failures of formal developments often result from

K. G. Larsen and T. Willemse (Eds.): FMICS 2019, LNCS 11687, pp. 1–17, 2019.
https://doi.org/10.1007/978-3-030-27008-7_6

misunderstandings of the users' needs or errors in the expression of these needs, although the system's correctness is entirely proved.

Petri-nets, introduced in 1962 [16], partially circumvent this shortcoming, since they combine a mathematical notation with an accessible graphical representation based on bipartite directed graphs. They are especially known to be powerful for event-driven systems [10] like distributed and real-time systems, logistic networks, embedded controllers, etc. Several high-level variants of Petri-nets, like coloured Petri-nets or predicate-transition nets, were applied in safety-critical systems and were assisted by formal verification techniques such as animation, model-checking or proofs. Moreover, some experiences like that of the Oslo subway, reported in [8], show that in addition to their formal semantics, high-level Petri-nets facilitated communication with domain experts, because chief engineers from railroad infrastructure and traffic department who are neither specialists in Petri-nets nor in formal methods, were not only able to understand the models, but also to suggest improvements.

Despite the Petri-nets advantages and their suitability for a readable formal description, their main disadvantage is that they miss out the system structure and focus on the system behaviour. Nonetheless, in the real world, structural and dynamic aspects of a system are often interdependent. For example, in train controlling systems the topology of the railroad, which defines position of track sections, orientation of switches and/or automatic train stopping devices with their corresponding signalling mechanisms, impacts the overall safety of train movements and behaviours. In safety critical systems, the system structure as well as critical situations that may arise from this structure are often provided by domain experts using informal graphical representations which may be referenced in the specification documents. We believe that these graphical representations should be defined in dedicated domain-specific languages (DSLs) with tool support, especially as modeling languages development is a well mastered technique today. The emergence of DSL tools in safety-critical systems [9,18,19] allows domain experts themselves to provide useful structural models to the software system engineer who will then develop the operational aspects of the system. However, as far as we know, none of the existing works in the safety-critical domain proposes a way to define proved formal links between the dynamic system description (in Petri-nets or other well known formalisms) and DSL tool development. There exist some attempts in model-driven engineering (MDE) with tools for executable DSLs [2,4,15], however they cannot be applied as is in safety critical systems because they are not assisted by automated reasoning tools and lack well-established verification and validation techniques.

This paper gives practical solutions to address this challenge starting from an intuitive description of a safety critical system where the dynamic aspects are specified thanks to high-level Petri-nets and the structural aspects are designed in a DSL tool. This work allows to enhance the usability of formal methods in industry because it involves domain experts all along the development process for both structural and behavioural modeling. Our approach uses the B method [1] in order to merge both worlds (that of Petri-nets and that of DSLs) and

then applies AtelierB in order to prove the correctness of the resulting static and dynamic semantics of the modelled system. We also apply the refinement principle of the B method to incrementally define formal operational semantics by means of refined Petri-net models. In every refinement step we introduce additional conceptual elements with associated safety properties and we prove the preservation of these properties as well as those of the previous level.

Section 2 gives the application context of this work. Sections 3 and 4 separately describe operational and static semantics and can be read in any order; and then, Sect. 5 puts it all together, using the B method, in Meeduse[1] – a tool that we developed in order to mix the formal B method with domain specific languages. Finally, Sect. 6 draws the conclusion and the perspectives of this work.

2 Application Context

This work is funded by the NExTRegio project of IRT Railenium. The project aims at performing a system level analysis of a railway signalling system taking into account emergent solutions for train automation. Indeed, in the last decade, new technologies have been considered in railway systems in order to improve automation on the one hand and to reduce the operating costs on the other hand. In particular, the European ERTMS/ETCS[2] [5,17] has emerged to replace various national signalling systems. There are three levels of ERTMS/ETCS which differ by the used equipments and the operating mode. The first two levels are already operational. However, ERTMS Level 3 is still in design and experimentation phases: it aims at replacing signalling systems with a global european one which is a GPS-based solution for the acquisition of train positions. In 2018, the ABZ conference [3], which gathers several formal methods communities, proposed a case study[3] to model ERTMS/ETCS level 3 and has published several formal models. Unfortunately, these models do not combine the power of formal methods with domain specific approaches and hence they favour verification ("do the system right") rather than domain expert validation ("do the right system"). The application presented in this paper contributes to the design phase of ERTMS/ETCS level 3 by mixing formal techniques and domain specific modeling in a well-known Model Driven Engineering (MDE) paradigm which makes easier domain expert validation without losing sight of the verification activity.

An ERTMS Level 3 solution is based on train position and train integrity confirmation, both transmitted by the on-board train system (called EVC[4]) to the trackside system (called RBC[5]). Given this information, the traffic agent, via RBC, assigns a movement authority to a train allowing it to move to a given

[1] http://vasco.imag.fr/tools/meeduse/.
[2] ERTMS: European Rail Traffic Management System.
ETCS: European Train Control System.
[3] https://www.southampton.ac.uk/abz2018/information/case-study.page.
[4] European Vital Computer.
[5] Radio Block Center.

point. In the RBC, track-circuits exist in a logical form by means of trackside train detection sections (called TTD) which are in turn divided into virtual sub-sections (called VSS). Figure 1, taken from the ERTMS 3 reference document [5], illustrates a track circuit divided into two TTDs and four VSSs, and where a train is located on VSS23. This simplified view of section conventions, used by railway experts, applies specific domain representations to represent a situation where a train went through TTD2 and reached its ending VSS.

The work proposed in this paper is intended to make domain specific models, provided by domain experts themselves, such as that of Fig. 1, not only provable but also executable thanks to the animation of their expected behaviour directly in the dedicated DSL tool. Operational semantics of these models are described using high-level Petri-nets, especially coloured Petri-nets (CP-nets), which abstract away structural constraints and focus on safety-critical behaviours. Static semantics of these models, together with their graphical representation, are developed in a MDE framework based on EMF[6] and Sirius[7].

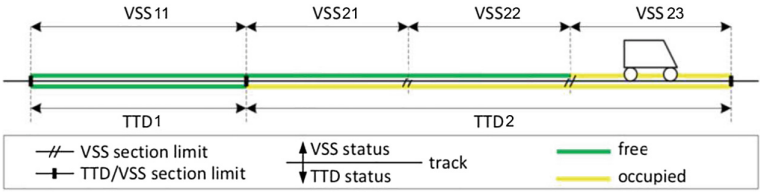

Fig. 1. Section conventions [5]

Figure 2 gives the overall architecture of the resulting models and formal specifications. The DSL meta-model and CP-Net models are automatically translated into B specifications which are enhanced by safety invariants and proved. Then, our approach defines linkage machines allowing to control the functional model and the associated DSL-tool thanks to the CP-Net specifications. Every linkage machine refines a CP-Net model and includes the functional model.

3 Coloured Petri-Nets: From Modeling to Proofs

3.1 Main Concepts

We use coloured Petri-nets (CP-nets) [12] because of their abstraction capabilities and their readability. They combine the strengths of classical Petri-nets with the strengths of high-level programming languages [7], to allow handling data types with pre-defined functions. For a formal description of CP-nets, one can refer to [11]; nonetheless, the main concepts used in this paper are:

[6] https://www.eclipse.org/modeling/emf/.
[7] https://www.obeo.fr/fr/produits/eclipse-sirius.

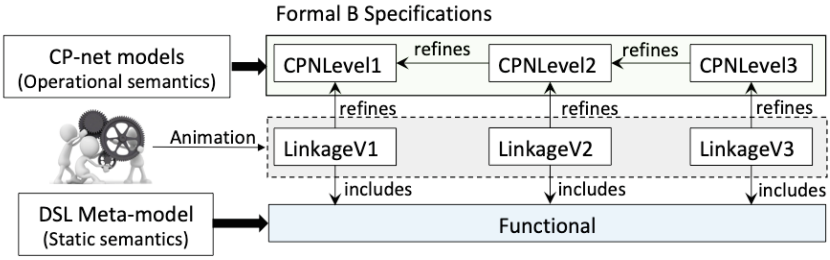

Fig. 2. Overall architecture of a formal DSML semantics

- Data types: can be simple types (*i.e.* Integer, Boolean,...) or complex types (*i.e.* arrays, sequences,...). In this work, we mainly use integer enumerations.
- Places: represent abstractions on data values (called tokens or colours). The place type is called the colour set and it is defined by composing data-types.
- Transitions: they are linked to input and output places. When fired, a transition consumes tokens from its input places such that they match the transition signature. Then, the transition introduces tokens into its output places.
- Predefined functions: describe some computations done by the transitions when they are fired. In this paper, we use three basic functions: calculation of the next (n^{++}) and the previous value (n^{--}) given a token n when n is of type integer, and the negation value ($\neg n$) when n is of type boolean.

3.2 Level 1: Simple Train Movements

Our first CP-net (Fig. 3) defines simple train movements without train integrity nor movement authorities. This abstract level is mainly intended to guarantee the absence of accidents.

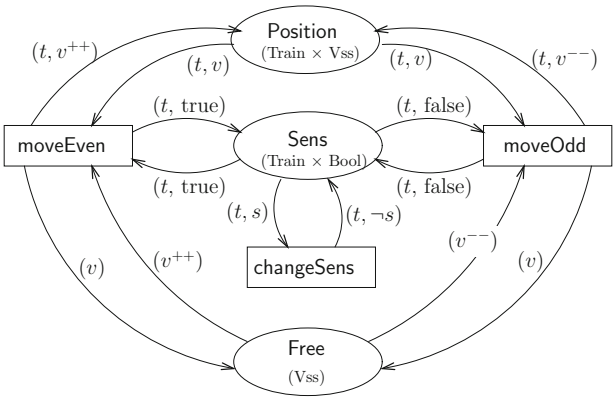

Fig. 3. Simple movement described in a coloured Petri-net

This model describes train movements using transitions moveEven and moveOdd which move the train forward or backward and; and changeSens which switches the train moving direction. Place Position contains pairs (t, v) which record the current VSS v occupied by a train t. Place Free gathers the sections which are not occupied by any train and place Sens registers for every train its current direction. For our first CP-net model, we would like to prove five safety properties:

1. Absence of accidents meaning that at most one train occupies a Vss,
2. Every train is located in one and only one Vss,
3. Absence of overlapping between Vss states free and occupied,
4. Vss states cannot be undefined, they are either free or occupied,
5. The train moving direction is never lost

Transition moveEven is fired given a train t located on section v, whose direction is set to $true$, and such that its next section v^{++} is free (*e.g.* $(t, v) \in Position \wedge (v^{++}) \in Free$). When fired, this transition instantly moves train t from section v to section v^{++}. It consumes tokens (t, v) and (v^{++}) respectively from places Position and Free, and then respectively introduces into these places tokens (t, v^{++}) and (v), meaning that v^{++} becomes the new position of train t, and section v is released. Transition moveOdd applies the same principles to trains in direction $false$ but selects the previous section v^{--} if this section is free.

3.3 Extraction of B Specifications

In order to prove the safety properties of our first level CP-net model we translate it into B specifications as follows:

MACHINE
CPNData
CONSTANTS
maxTrain, *maxVss*, *CPNTrain*, *CPNVss*
PROPERTIES
$maxTrain \in \mathbf{NAT} \wedge maxVss \in \mathbf{NAT}$
$\wedge\ CPNTrain = 1\,..\,maxTrain$
$\wedge\ CPNVss = 1\,..\,maxVss$

REFINEMENT *CPNLevel1*
REFINES *CPNData*
VARIABLES
Free, *Position*, *Sens*
INVARIANT
$Free \subseteq CPNVss$
$\wedge\ Position \subseteq CPNTrain \times CPNVss$
$\wedge\ Sens \subseteq CPNTrain \times \mathbf{BOOL}$

First an abstract machine (named CPNData) is generated in order to gather the colour sets together with the transition signatures as defined in the CP-net model. Colour sets Train and Vss, which are integer enumerations, are translated into bounded natural constants CPNTrain and CPNVss. Places Free, Position and Sens become variables in refinement CPNLevel1 because their values evolve during the execution of the CP-net. In this refinement, by default the variable typing applies general functions such as sets' cartesian product and inclusion (*e.g.* $Position \subseteq CPNTrain \times CPNVss$).

Every transition leads to a basic operation defined in machine CPNData with a typing precondition and a skip substitution, like the example below of operation moveEven:

```
/* Operation moveEven in machine CPNData */
moveEven(tt, vv) =
    PRE tt ∈ CPNTrain ∧ vv ∈ CPNVss THEN
        skip
    END
```

The skip substitution of the basic operations is then refined in CPNLevel1 by introducing the enabledness guards and the expected actions of the transition. In the following we give the refinement of operation moveEven in CPNLevel1:

```
/* Refinement of the skip substitution in CPNLevel1 */
moveEven(tt, vv) =
    SELECT
        (tt ↦ vv) ∈ Position ∧ (vv + 1) ∈ Free ∧ (tt ↦ TRUE) ∈ Sens
    THEN
        Free := (Free − {(vv + 1)}) ∪ {(vv)} ||
        Position := (Position − {(tt ↦ vv)}) ∪ {(tt ↦ vv + 1)}
    END ;
```

Transitions moveOdd and changeSens are translated by applying the same principles. Regarding the five safety properties, they are manually introduced in machine CPNLevel1 using the following invariants:

```
Position ∈ CPNTrain ⤔ CPNVss /* Properties (1) and (2) */
Free ∩ ran(Position) = ∅ /* Property (3) */
Free ∪ ran(Position) = CPNVss /* Property (4) */
Sens ∈ CPNTrain → BOOL /* Property (5) */
```

These invariants restrict the state space defined by the typing predicates presented above. For example, the typing predicate of relation *Position* defines all combinations of CPNTrain and CPNVss couples, while the invariant restricts these combinations to those where a CPNTrain is linked to one and only one CPNVss while a CPNVss is linked to at the most one CPNTrain. In our methodology, we consider that if the CP-net model is correct, proofs should be done without any enhancement of the corresponding B specifications. Otherwise, we decide whether the CP-net model is wrong or not, given the AtelierB feedbacks. In all cases we do not modify the generated B operations; we either call the interactive prover when the proof fails due to a limitation in the automatic prover, or we correct the CP-net model and translate it again into B. The initial marking substitutions are introduced without invariant violation:

```
INITIALISATION
    Position :∈ CPNTrain ⤔ CPNVss ;
    Free := CPNVss − ran(Position) ;
    Sens :∈ CPNTrain → BOOL
```

Based on machines CPNData and CPNLevel1, and these additional invariants, the AtelierB generated 17 proof obligations and automatically proved 11 amongst them. The 6 other POs were proved using the interactive prover.

4 A Railway Domain-Specific Modeling Language

4.1 Railway Meta-Model

In order to provide a tool for domain experts allowing them to draw models like that of Fig. 1, we apply model-driven engineering tools for DSML creation (EMF, Ecore-Tools and Sirius). In MDE, the creation of a DSML starts by the definition of its meta-model and then for every class in the meta-model a graphical representation is created. Figure 4 gives the meta-model that we use in this work and Fig. 5 gives a screenshot of the resulting DSML-tool in which a model is designed using the proposed graphical representations.

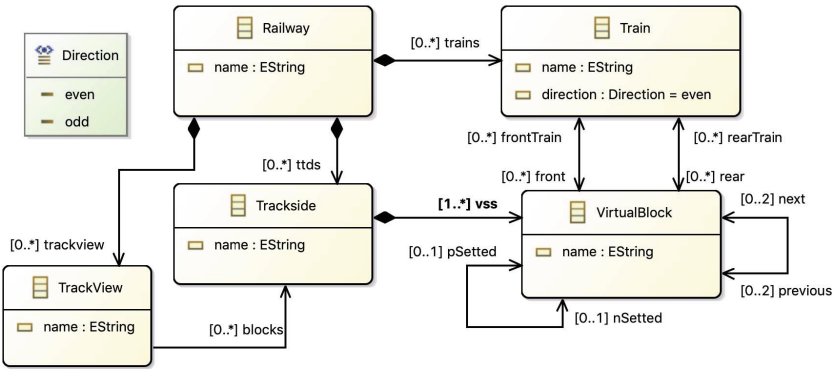

Fig. 4. A railway meta-model

In our meta-model, a railway system is composed of trains (class Train), track sections called TTD in ERTMS/ETCS 3 (class Trackside) and which are divided into portions called VSS (class VirtualBlock). The bottom of Fig. 5 draws an overall railway topology by means of TTD links. Every portion of a given TTD may be linked to two next and previous portions at the most. In practice, there are four kinds of portions: track extremity (*e.g.* VSS11 and VSS62), middle track (*e.g.* VSS12), switch (*e.g.* VSS21 and VSS51) and diamonds. Association pSetted/nSetted provides the currently selected previous/next portion among those to which a portion is linked. This is useful especially for switches and diamonds. For example, the next portions of VSS21 are VSS31 and VSS41, but the position of the switch sets the currently selected next portion of VSS21 to VSS31 and hence the selected previous portion of VSS31 is VSS21 but for portion VSS41 there is no previous selected portion. Portion VSS41 remains then a track limit until the switch position is changed. Note that relation pSetted/nSetted is independent from train direction and a track limit is a portion without a selected next or previous portion.

Class TrackView represents linear views that follow the current next/previous selections and where every view starts and ends with track limits. For example,

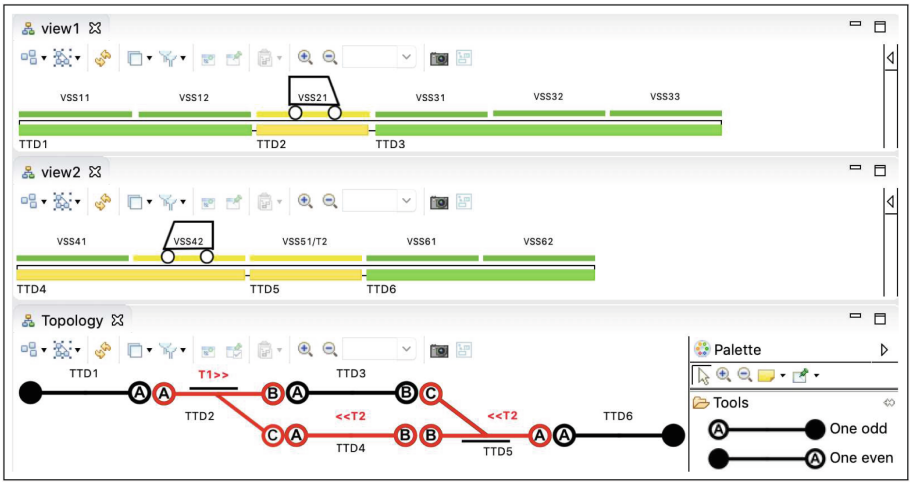

Fig. 5. A railway model (Color figure online)

the topology presented in the bottom of Fig. 5, leads to the two views on the top of the figure. The first view covers sections TTD1, TTD2 and TTD3 and the second view covers the three other sections: TTD4, TTD5 and TTD6. If the switches position changes, these views are changed consequently. For example, if the selected next portion of VSS21 is set to VSS41, then the resulting topology would lead to two different views: one composed of TTD1/TTD2/TTD4/TTD5/TTD6, and an other view dedicated to TTD3 only.

Trains have a direction (even or odd) and their representation depends on the set of portions that their head and rear occupy. In the example of Fig. 5 we consider two trains: T1 whose front and rear occupy the same portion (*i.e.* VSS21), and T2 that stretches from portion VSS42 to VSS51. A TTD is occupied when at least one of its portions are occupied. This is represented by the yellow color in the track views and by the red color in the topology representation. The green color is used to represent free TTD and VSS in the track view.

4.2 Formal Model

As our intention is to provide domain experts with a DSML-tool with formal semantics, we apply the Meeduse platform[8] that we developed in order to automatically translate a meta-model into an equivalent B specification. The resulting formal model gathers the structure of the meta-model (by means of sets, variables and structural invariants) with a set of basic operations such as constructors, getters and setters. For example, we give below the translation of classes Train and VirtualBlock and one basic operation Train_AddFront which adds a virtual block to the set of virtual blocks occupied by the head of a train.

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

Several other basic operations are generated by the tool like: Train_RemoveFront, Train_AddRear, Train_RemoveRear...

MACHINE *Functional*
SETS
VIRTUALBLOCK; *TRACKSIDE*
Direction = {*even*,*odd*};
VARIABLES
Train, *VirtualBlock*, *Train_direction*,
frontOfTrain, *rearOfTrain*
INVARIANT
Train ⊆ *TRAIN*
∧ *VirtualBlock* ∈ *VIRTUALBLOCK*
∧ *frontOfTrain* ∈ *Train* ↔ *VirtualBlock*
∧ *rearOfTrain* ∈ *Train* ↔ *VirtualBlock*
∧ *Train_direction* ∈ *Train* → *Direction*

Train_AddFront(*aTrain*,*aFront*) =
PRE
aTrain ∈ *Train* ∧
aFront ∈ *VirtualBlock* ∧
(*aTrain* ↦ *aFront*) ∉ *frontOfTrain*
THEN
frontOfTrain :=
frontOfTrain ∪ {(*aTrain* ↦ *aFront*)}
END;

The translation of a meta-model into B applies a UML-to-B transformation technique where a meta-class Class is translated into 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.* Direction). Basic types (*e.g.* integer, boolean) become B types (Z, Bool,...). Attributes and references lead to functional relations depending on multiplicities.

Machine Functional generated by Meeduse is about 500 lines with 38 basic operations from which the AtelierB produced 80 proof obligations that were proved automatically. Proofs associated to this functional specification guarantee that the basic operations do not violate the structural properties of the meta-model such as multiplicities and single-valued and mandatory attributes, etc. Besides the automatic extraction of a correct by-design functional B specification, the interest of Meeduse is that it integrates the ProB [14] animator. Given a model (like that of Fig. 5) Meeduse injects it as valuations in the B specification and calls ProB in order to compute the list of operations that may be animated from these valuations. For example, the following initialization is extracted by Meeduse from our graphical model which leads to an initial state of the B machine which is conformant with the domain model.

INITIALISATION
Train := {*T1*, *T2*} ||
VirtualBlock := {*VSS11*, *VSS12*, ..., *VSS62*} ||
frontOfTrain := {(*T1* ↦ *VSS21*), (*T2* ↦ *VSS42*)} ||
rearOfTrain := {(*T1* ↦ *VSS21*), (*T2* ↦ *VSS51*)} ||
Train_direction := {(*T1* ↦ *even*), (*T2* ↦ *odd*)}

Starting from the initial state, when the user asks Meeduse to animate a B operation, the tool calls ProB and gets the new variable valuations and then it translates back these valuations to the graphical model. This technique results in

an automatic visual animation[9] of domain models. For example, given the above initial state, the animation of operation Train_AddFront(T1, VSS31) introduces couple ($T1 \mapsto VSS31$) into relation *frontOfTrain* and then Meeduse modifies the domain model as presented in Fig. 6 where the head of T1 occupies two virtual blocks VSS21 and VSS31. Since VSS31 is one of the portions of TTD3, then the visual representation of TTD3 automatically changes from green to yellow.

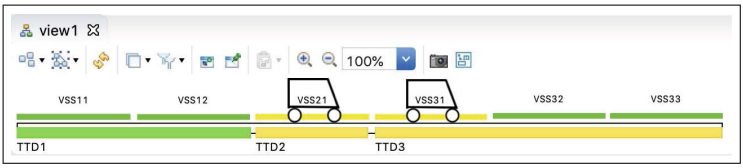

Fig. 6. View 1 after animation of Train_AddFront(T1, VSS31) (Color figure online)

5 Putting It All Together

Section 3 focused on train behaviours with an abstract Petri-net specification that guarantees the absence of accidents, and Sect. 4 focused on domain modeling of structural aspects of a railway DSML. In this section, we combine both concerns in order to provide a railway DSML with a proved safe train behaviour. The B specifications extracted from the meta-model of Fig. 4 represent formal static semantics of our DSML, and those extracted from a coloured Petri-net model introduce its operational semantics. In order to merge static and operational semantics we create machine LinkageV1 which refines CPNLevel1 and includes machine Functional:

REFINEMENT *LinkageV1*
REFINES *CPNLevel1*
INCLUDES *Functional*
VARIABLES
trainMapping, *vssMapping*, *view*
INVARIANT
$trainMapping \in Train \rightarrowtail\!\!\!\!\!\!\rightarrow CPNTrain$
$\wedge\ vssMapping \in VirtualBlock \rightarrowtail\!\!\!\!\!\!\rightarrow CPNVss$
$\wedge\ view \in TrackView$

The refinement guarantees the preservation of the safety invariants of CPNLevel1 and the inclusion allows to redefine the Petri-net transitions and data using the functional variables of the DSML. In this machine the linkage between the DSML and the CP-net model is done via functions *trainMapping* and *vssMapping*. They respectively map variables *Train* and *VirtualBlock* issued from the

[9] For place reason we do not develop the animation technique in this paper. Demonstration videos of Meeduse with graphical and textual DSL animation can be found at: http://vasco.imag.fr/tools/meeduse/.

meta-model to sets *CPNTrain* and *CPNVss* issued from the CP-net. In our approach every view in the DSML is controlled by a CP-net since the CP-net defines the VSS set by a sequence of integers. Then, the mapping functions are applied to a given *view* (*view* ∈ *TrackView*). For example, the *vssMapping* relation is computed in the initialisation of LinkageV1 as:

LET $mapVss$ **BE** $mapVss = \mathbf{ran}((\{view\} \lhd blocks^{-1} ; theVSSs^{-1}))$ **IN**
  ANY *map* **WHERE**
    $map \in mapVss \rightarrowtail CPNVss \wedge$
    $\forall\, vss \,.\, (vss \in mapVss \wedge nSetted[\{vss\}] \neq \emptyset$
      $\Rightarrow nSetted(vss) \in \mathbf{dom}(map) \wedge map(nSetted(vss)) = map(vss) + 1)$
  THEN
    $vssMapping := map$
  END
END

Note that *blocks* and *theVSSs* represent respectively association blocks between classes TrackView and Trackside, and association vss between classes Trackside and VirtualBlock. Local variable *mapVss* defined by: $\mathbf{ran}((view \lhd blocks^{-1} ; theVSSs^{-1}))$ extracts the set of VSS for a given *view* and the mapping is a total injection ($\rightarrowtail$) that maps every VSS in this view to a unique value from set *CPNVss*. This mapping is done under the condition that if a VSS is not a track extremity ($nSetted[vss] \neq \emptyset$) then its next selected VSS is mapped ($nSetted(vss) \in \mathbf{dom}(map)$) and the associated CP-net value is equal to the VSS value plus one. We similarly compute the *trainMapping* relation but under the condition that only trains whose head and rear occupy the same VSS are mapped. In this sense, from the example of Fig. 5 only the first view can be mapped and then controlled by our first level CP-net model.

Given the mapping relations, the safety invariants of CPNLevel1 are rewritten by means of linkage invariants ensuring the relationship between the various B specifications. For example, invariant $Free \cap \mathbf{ran}(Position) = \emptyset$ used for `Property (3)` becomes:

$$(frontOfTrain \cup rearOfTrain)^{-1}[vssMapping^{-1}[Free]] = \emptyset$$

which means that for every free VSS in the CP-net model, the corresponding virtual block in the DSML does not contain any train head or rear. Having the linkage invariants, operation moveEven(tt, vv) in the linkage machine is applied to a train mapped to *tt*, whose head and rear occupy a VSS mapped to *vv*, and whose direction is *even* and such that the next VSS which is mapped to $vv + 1$ is free. Actions of moveEven call basic functional operations issued from machine Functional. They simply remove the head and the rear of the train from *vv* and put them on $vv + 1$. In the following we give the refinement of operation moveEven in LinkageV1:

moveEven(tt, vv) =
LET $train$, vss, $nextVss$ **BE**
$train = trainMapping^{-1}(tt)$
$\wedge\ vss = vssMapping^{-1}(vv)$
$\wedge\ nextVss = vssMapping^{-1}(vv + 1)$
IN
SELECT
$(train \mapsto vss) \in frontOfTrain \cap rearOfTrain$
$\wedge\ nextVss \notin \mathbf{ran}(frontOfTrain \cup rearOfTrain)$
$\wedge\ Train\_direction(train) = even$
THEN
Train_RemoveFront($train$, vss); **Train_AddFront**($train$, $nextVss$) ;
Train_RemoveRear($train$, vss); **Train_AddRear**($train$, $nextVss$)
END
END ;

At this stage we are able to do verification and validation. Indeed, verification is done thanks to the 41 POs that were proved by the AtelierB for machine LinkageV1 and which mean that the safety properties (those of CPNLevel1) as well as the structural properties (those of Functional) are preserved. Regarding validation, it is done by railway experts using the animation facility of Meeduse. As we showed previously, Meeduse animation of B operations that impact the functional model automatically animates the corresponding graphical model.

5.1 Incremental Development of Operational Semantics

CPNLevel1 describes simple train movements without train integrity nor movement authorities which are basic concepts of ERTMS/ETCS 3. This specification guarantees the absence of accidents and defines a first abstraction level of our DSML operational semantics. In this section, we show how operational semantics, can be incrementally defined in order to first introduce movement authorities and then the track release mechanism when the train integrity is confirmed.

Machines CPNLevel1, Functional and LinkageV1 of Fig, 2 were discussed in the previous sections. Machines CPNLevel2 and CPNLevel3 are extracted from additional CP-net models and apply a refinement technique where every refinement level introduces new safety properties without violating the properties of the previous levels. In this section we mainly discuss CP-net refinements. Machines LinkageV2 and LinkageV3 will not be discussed since they are defined via the same principles as LinkageV1. They allow the domain expert to animate the domain model for every CP-net refinement and validate the observed behaviours. Thanks to these machines, the domain expert is involved all along the development process.

Level 2: Authorized Train Movements

The assumption made in the first CP-net level, considering that a train moves to the next free virtual section and immediately leaves its current section, is quite

simplistic but sufficient in order to model an abstract accident-free behaviour. In this second level we introduce a movement authority mechanism, in order to construct routes to which trains are allowed to move. The movement authority, in the ERTMS/ETCS, is used without visual signals or marker boards. It is sent by the RBC system to a given train via GSM-R.

Our objective is to prove that authorized train movements, no matter how authorizations are assigned to trains, preserve the accident-free behaviour of the previous level. Figure 7 is a CP-net model which includes authorized movements and where we focus on the refinement of transition moveEven and state Free. In addition to transition moveOdd which is analog to moveEven, and transition changeSens which is kept unchanged, this model introduces transition authorize which represents the actions executed by a train when it receives a movement authority signal from the RBC.

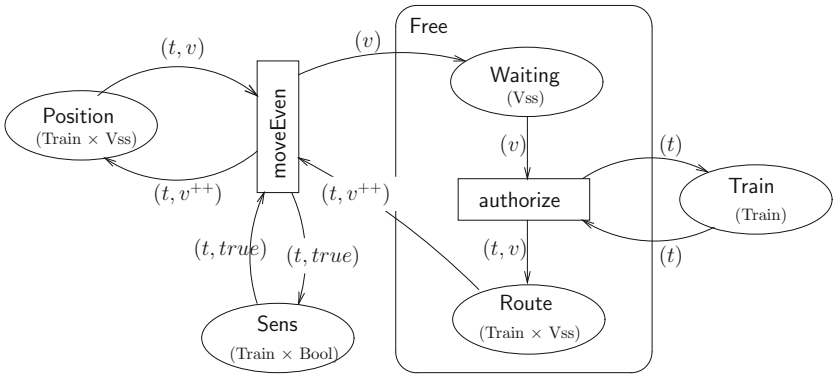

Fig. 7. CP-net for authorized train movements

In this CP-net model, place Free is refined into two places: Waiting and Route. When a train moves away from a given Vss, the Vss is freed but cannot be used before being reserved. The Vss first enters in place Waiting and then transition authorize assigns it to a given train and adds the corresponding movement authority to place Route. The extraction of B specifications follows the same principles as discussed for Level 1, and produces the variables with their typing invariants showed below:

REFINEMENT *CPNLevel2*
REFINES *CPNLevel1*
VARIABLES
Position, *Sens*, *Waiting*, *Route*
INVARIANT
`/* Typing invariant generated from CP-net model */`
$Waiting \subseteq CPNVss \wedge Route \subseteq CPNTrain \times CPNVss$
`/* Refinement invariant */`
$Waiting \cup \mathbf{ran}(Route) = Free$

```
/* Refinement of moveEven with authorized movements */
moveEven(tt,vv) =
    SELECT
        (tt ↦ vv) ∈ Position ∧ (tt ↦ vv + 1) ∈ Route ∧ (tt ↦ TRUE) ∈ Sens
    THEN
        Position := (Position − {(tt ↦ vv)}) ∪ {(tt ↦ vv + 1)} ||
        Route := Route − {(tt ↦ vv + 1)} ||
        Waiting := Waiting ∪ {vv}
    END ;
```

The refinement invariant means that the set of tokens of place Free are distributed among places Waiting and Route, and then variable Free is replaced by variables Waiting and Route which are used in the refinement of transition moveEven. The additional safety invariants of this second CP-net level are: (6.) a VSS cannot be waiting and at the same time assigned to a movement authority; and (7.) a movement authority cannot be shared by several trains.

```
Waiting ∩ ran(Route) = ∅ /* Property (6) */
Route ⁻¹ ∈ CPNVss ⇸ CPNTrain /* Property (7) */
```

Given CP-net of Level 2 and the corresponding safety properties, as well as the refinement invariant, the AtelierB prover generated 32 POs, such that 25 were proved automatically and 7 interactively, which means that CP-net Level 2 guarantees its own properties and also those of CP-net Level 1.

Level 3: Movements with Integrity Confirmation

In the third refinement level we consider a more realistic train representation than that developed in the two previous levels where a train occupies only one VSS. In this refinement, a train is seen as a logical entity defined by the set of VSS that it occupies: its head (place Position), a set of VSS not yet released behind its head (place Wagon) and the safe rear end (place Tail) which is in our case one additional VSS defining the minimal distance between two trains. Thus, a train occupies at least two virtual sections: one for its head and one behind it. When a train moves, its head is advanced from its current VSS v to the next VSS v^{++}, and then v is not freed but a virtual wagon is created over it. Indeed, in ERTMS/ETCS 3, the train must confirm its integrity (*i.e.* it did not lose wagons) before releasing its safe rear end which advances its tail by one VSS and removes the corresponding virtual wagon. Figure 8 provides the refinement of CP-net level 2 introducing integrity confirmation together with the VSS release mechanism. This model introduces places Wagon, Tail and Ready as a refinement of place Waiting, and transition confirmEven which is fired when a train integrity is confirmed. A released VSS becomes ready for reservation and enters in place Ready. Given the B specifications issued from this third level and the associated safety invariants, the AtelierB produced 62 POs and automatically proved 41 among them. The 21 other POs were proved manually.

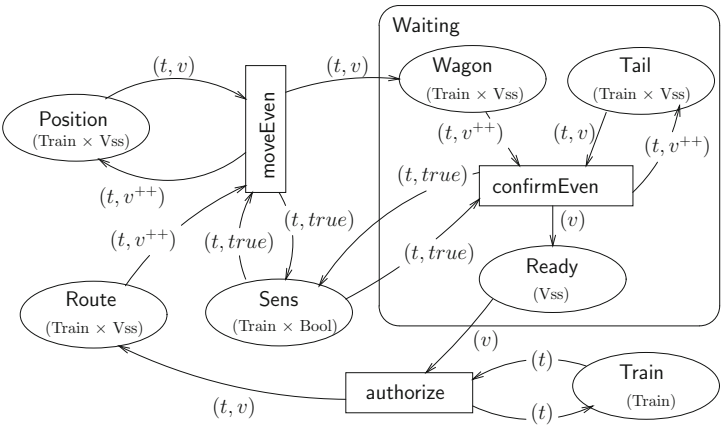

Fig. 8. CP-net for authorized movements and integrity confirmation

6 Conclusion

This paper presented an incremental formal development process that involves domain experts during the modeling activities. First, we use coloured Petri-nets because their graphical notations are more readable than textual mathematical notations. Then, we use DSMLs in order to assist domain experts for the design of the domain models. CP-nets and DSMLs are convenient for validation because they both favour communication complementing each other by focusing on particular concerns: behavioural concerns for CP-nets, and structural concerns for DSMLs. In order to mix the various models we apply the B method which allows a proof-based verification thanks to the AtelierB prover, and domain model animation thanks to Meeduse and ProB. Our approach was successfully applied to a railway safety critical system, the ERTMS/ETCS 3 train automation solution and other case studies (automatic car light regulator, parking-lot controller,. . .).

Several perspectives araise from this work, especially we plan to develop an automated extraction technique of sub-parts of the linkage machines. In this work, these machines were introduced manually which is still somehow difficult and time consuming when the CP-net scales up such as our third CP-net refinement. We also plan an empirical study with railway experts in order to validate the usability of our tool-set. The validation is currently limited to academic railway experts of the NExTRegio project.

References

1. Abrial, J.-R.: The B-book: Assigning Programs to Meanings. Cambridge University Press, New York (1996)
2. Bousse, E., Leroy, D., Combemale, B., Wimmer, M., Baudry, B.: Omniscient debugging for executable DSLs. J. Syst. Softw. **137**, 261–288 (2018)

3. Butler, M., Raschke, A., Hoang, T.S., Reichl, K. (eds.): ABZ 2018. LNCS, vol. 10817. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91271-4
4. Deantoni, J.: Modeling the behavioral semantics of heterogeneous languages and their coordination. In: 2016 Architecture-Centric Virtual Integration (ACVI), pp. 12–18, April 2016
5. EEIG ERTMS USERS GROUP ERA, UNISIG. System Requirements Specification, SUBSET-026. Technical report, European Railway Agency, Version 3.6.0 (2016)
6. Gaudel, M.-C.: Advantages and limits of formal approaches for ultra-high dependability. In: Randell, B., Laprie, J.C., Kopetz, H., Littlewood, B. (eds.) Predictably Dependable Computing Systems. ESPRIT Basic Research Series, pp. 241–251. Springer, Heidelberg (1995). https://doi.org/10.1007/978-3-642-79789-7_14
7. Gehlot, V., Nigro, C.: An introduction to systems modeling and simulation with colored petri nets. In: Proceedings of the 2010 Winter Simulation Conference, WSC 2010, USA, 5–8 December 2010, pp. 104–118 (2010)
8. Hagalisletto, A.M., Bjørk, J., Yu, I.C., Enger, P.: Constructing and refining large-scale railway models represented by petri nets. IEEE Trans. Syst. Man Cybern. Part C **37**(4), 444–460 (2007)
9. 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
10. Janczura, C.: Modelling and analysis of railway network control logic using coloured Petri Nets. Ph.D. thesis. University of South Australia (1998)
11. Jensen, K.: Coloured Petri Nets and the invariant-method. Theor. Comput. Sci. **14**, 317–336 (1981)
12. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, vol. 1. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-662-03241-1
13. Lecomte, T.: Applying a formal method in industry: a 15-year trajectory. In: Alpuente, M., Cook, B., Joubert, C. (eds.) FMICS 2009. LNCS, vol. 5825, pp. 26–34. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04570-7_3
14. Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. STTT **10**(2), 185–203 (2008)
15. Mayerhofer, T., Langer, P., Wimmer, M., Kappel, G.: xMOF: executable DSMLs based on fUML. In: Erwig, M., Paige, R.F., Van Wyk, E. (eds.) SLE 2013. LNCS, vol. 8225, pp. 56–75. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02654-1_4
16. Petri, C.-A.: Fundamentals of a theory of asynchronous information flow. In: IFIP Congress, pp. 386–390 (1962)
17. Schn, W., Larraufie, G., Mons, G., Por, J.: Railway signalling and automation, vol. 3. La vie du rail (2014)
18. 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
19. 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 (2014)