Modeling the Hybrid ERTMS/ETCS Level 3 Standard Using a Formal Requirements Engineering Approach

Steve Jeffrey Tueno Fotso[1,2], Marc Frappier[1], Régine Laleau[2], and Amel Mammar[3]

[1] Université de Sherbrooke, GRIL, Québec, Canada,
`Steve.Jeffrey.Tueno.Fotso@USherbrooke.ca`, `Marc.Frappier@USherbrooke.ca`
[2] Université Paris-Est Créteil, LACL, Créteil, France,
`laleau@u-pec.fr`
[3] Télécom SudParis, SAMOVAR-CNRS, Evry, France,
`amel.mammar@telecom-sudparis.eu`

Abstract. This paper presents a specification of the hybrid ERTMS/ETCS level 3 standard in the framework of the case study proposed for the 6th edition of the ABZ conference. The specification is based on the method and tools, developed in the *ANR FORMOSE* project, for the modeling and formal verification of critical and complex system requirements. The requirements are specified with *SysML/KAOS* goal diagrams and are automatically translated into *B System* specifications, in order to obtain the architecture of the formal specification. Domain properties are specified by ontologies with the SysML/KAOS domain modeling language, based on *OWL* and *PLIB*. Their automatic translation completes the structural part of the formal specification. The only part of the specification, which must be manually completed, is the body of events. The construction is incremental, based on the refinement mechanisms existing within the involved methods. The formal specification of the case study is composed of seven refinement levels and all the proofs have been discharged with the Rodin prover.

Keywords: Requirements Engineering, Goal Diagrams, Domain Modeling, Ontologies, *SysML/KAOS*, *B System*

1 Introduction

In this paper, we are interested in using the *FORMOSE* approach [2] on the case study proposed for the 6th edition of the ABZ conference [7]. This case study deals with the specification of the *hybrid ERTMS/ETCS level 3* standard [5,12]. The case study is described in two main documents. The first one, [7], describes the hybrid ERTMS/ETCS level 3 protocol in a general way and restricts the scope of the study. The second one, [5], offers a technical and detailed description of the protocol specification. It provides the safety requirements that the system must guarantee. The FORMOSE method includes the *SysML/KAOS* requirements engineering language [6,10] for modeling requirements with goal diagrams. Domain properties are modeled with ontologies, using the SysML/KAOS domain modeling language [18,16]. Once done, translation rules [11,19,17], supported by tools [11,19], allow the automatic generation of the *B system* specification. The goal diagrams give the set of *B System* components, each goal gives an event. As the refinement links defined between these components have to represent the SysML/KAOS refinements, new proof obligations are generated. The domain model gives the structural part of the specification. It consists of variables, constrained by an invariant, and constants, constrained by properties. The Rodin tool [3] has been used to verify and validate the formal specification, especially to prove the safety invariants and the refinement logic. The complete specification can be found in [19].

The remainder of this paper is structured as follows: Section 2 briefly describes the *B System* formal method, the SysML/KAOS goal and domain modeling languages and the rules for obtaining the *B System* specifications. Follows a presentation, in Section 3, of the work done on the case study and in Section 4, of the discussion related to it. The discussion includes a short comparison with a companion paper on the same case study, but specified using only plain *Event-B*. Finally, Section 5 reports our conclusions.

2 Context

2.1 B System

Event-B is an industrial-strength formal method for *system modeling* [1]. It is used to incrementally construct a system specification, using refinement, and to prove properties. Proof obligations are defined to prove invariant preservation by events (invariant has to be true at any system state), event feasibility, convergence and machine refinement [1]. *B System* is an *Event-B* syntactic variant proposed by *ClearSy*, an industrial partner in the *FORMOSE* project [2], and supported by *Atelier B* [4]. A *B System* specification consists of components. Each component can be either a system or a refinement and it may define static or dynamic elements. Constants, abstract and enumerated sets, and their properties, constitute the static part. The dynamic part includes the representation of the system state using variables constrained through invariants and updated through events. Each event has a ***guard*** and an ***action***. The *guard* is a condition that must be satisfied for the event to be triggered and the *action* describes the update of state variables. Although it is advisable to always isolate the static and dynamic parts of the *B System* formal model, it is possible to define the two parts within the same component. In the following sections, our *B System* models will be presented using this facility.

2.2 SysML/KAOS Goal Modeling

SysML/KAOS [6,10] is a requirements engineering method which extends the *SysML* UML profile with a set of concepts from KAOS [8] allowing to represent functional and non-functional requirements.It combines the traceability features provided by *SysML* with goal expressiveness provided by *KAOS*. SysML/KAOS goal models allow the representation of requirements to be satisfied by the system and of expectations with regard to the environment through a goal hierarchy. The hierarchy is built through a succession of refinements using different operators: ***AND***, ***OR*** and ***MILESTONE***. An ***AND*** *refinement* decomposes a goal into subgoals, and all of them must be achieved to realise the parent goal. An ***OR*** *refinement* decomposes a goal into subgoals such that the achievement of only one of them is sufficient for the accomplishment of the parent goal. A ***MILESTONE*** **refinement** is a variant of the AND refinement which allows the definition of an achievement order between goals. A SysML/KAOS goal can be *functional* or *non-functional*. The scope of this document is limited to functional goals. A functional goal describes the *expected behaviour* of the system once a certain condition holds [10] : *[if* ***CurrentCondition*** *then] sooner-or-later* ***TargetCondition***. SysML/KAOS allows the definition of a functional goal without specifying a current condition. In this case, the expected behaviour can be observed from any system state.

2.3 Formalisation of SysML/KAOS Goal Models

The formalisation of SysML/KAOS goal models is the focus of the work done by [11]. The proposed rules allow the generation of a formal model whose structure reflects the hierarchy of the SysML/KAOS goal diagram : one component is associated with each hierarchy level; this component defines one event for each goal. The semantics of refinement links between goals is expressed in the formal specification with a set of proof obligations which complement the standard proof obligations for *invariant preservation* and for *event actions feasibility* [1]. Regarding the new proof obligations, they depend on the goal refinement operator used. For an abstract goal G and two concrete goals G_1 and G_2 : [4]

- For the *AND* operator, the proof obligations are
  - $G_1\_Guard \Rightarrow G\_Guard$
  - $G_2\_Guard \Rightarrow G\_Guard$
  - $(G_1\_Post \wedge G_2\_Post) \Rightarrow G\_Post$
- For the *OR* operator, they are
  - $G_1\_Guard \Rightarrow G\_Guard$
  - $G_2\_Guard \Rightarrow G\_Guard$
  - $G_1\_Post \Rightarrow G\_Post$
  - $G_2\_Post \Rightarrow G\_Post$
  - $G_1\_Post \Rightarrow \neg G_2\_Guard$
  - $G_2\_Post \Rightarrow \neg G_1\_Guard$
- For the *MILESTONE* operator, they are
  - $G_1\_Guard \Rightarrow G\_Guard$
  - $G_2\_Post \Rightarrow G\_Post$
  - $\Box(G1\_Post \Rightarrow \Diamond G2\_Guard)$ (each system state, corresponding to the post condition of G_1, must be followed, at least once in the future, by a system state enabling G_2)

Nevertheless, the generated specification does not contain the system structure, composed of variables, constrained by an invariant, and constants, constrained by properties.

[4] For an event `G`, `G_Guard` represents the guards of G and `G_Post` represents the post condition of its actions.

2.4 SysML/KAOS Domain Modeling

The SysML/KAOS domain modeling language [18,16] uses ontologies to represent domain models. It is based on *OWL* [15] and *PLIB* [14], two well-known ontology modeling languages. Each domain model corresponds to a refinement level in the SysML/KAOS goal model. The *parent* association represents the hierarchy of domain models. A domain model can define multiple elements. For this case study, a domain model can define concepts, attributes, datasets and predicates. A concept represents a collection of individuals with common properties. It can be declared variable ($isVariable = TRUE$) when the set of its individuals can be dynamically updated by adding or deleting individuals. Otherwise, it is constant ($isVariable = FALSE$). A data set represents a collection of data values. An attribute captures links between concepts and data sets. It can be variable or constant, functional or total. A predicate expresses constraints between domain model elements, using the first order logic. Each predicate has a body which represents its antecedent and a head which represents its consequent. The head can be omitted if it is always TRUE. Gluing invariants represent links between variables defined within a domain model and those appearing in more abstract domain models. They are extremely important because they capture relationships between abstract and concrete data during refinement and are used to discharge proof obligations.

2.5 From SysML/KAOS Domain Models to B System Specifications

The translation rules are fully described in [19,17]. They allow the extraction of the structural part of the system formal specification from domain models. Table 1 represents some rules that are relevant for our purposes. It should be noted that *o_x* designates the result of the translation of x and that *abstract* is used for "without parent".

Table 1: Summary of the translation rules

	Domain Model		**B System**	
	Element	**Constraint**	**Element**	**Constraint**
Abstract domain model	DM	$DM \in$ DomainModel DM is not associated to a parent domain model	o_DM	$o\_DM \in$ System
Domain model with parent	DM PDM	$\{DM, PDM\} \subseteq$ DomainModel DM is associated to PDM through the *parent* association PDM has already been translated	o_DM	$o\_DM \in$ Refinement o_DM refines o_PDM
Abstract concept	CO	$CO \in$ Concept CO is not associated to a parent concept	o_CO	$o\_CO \in$ AbstractSet

Attribute	AT CO DS	$CO \in$ Concept
		$DS \in$ DataSet
		$AT \in$ Attribute
		CO is the *domain* of AT
		DS is the *range* of AT
		CO and DS have already been translated
			o_AT	**IF** the $isVariable$ property of AT is set to $FALSE$
				THEN $o\_AT \in$ **Constant**
				ELSE $o\_AT \in$ **Variable**
				END
				$o\_AT \in o\_CO \leftrightarrow o\_DS$[5]

Data value	Dva DS	$Dva \in$ DataValue $DS \in$ DataSet Dva is a value of DS DS has already been translated	o_Dva	$o\_Dva \in$ Constant $o\_Dva \in o\_DS$

[5] Depending on attribute properties, this relation may become a partial or total function.

3 Specification of the Hybrid ERTMS/ETCS Level 3 Standard

3.1 Main Characteristics of the Standard

The Hybrid ERTMS/ETCS level 3 protocol (HEEL3) has been proposed to optimize the use and occupation of railways [7,5,12]. It thus proposes the division of the track into separate entities, each named Trackside Train Detection (TTD). In addition, each TTD is subdivided into sub-entities called Virtual Sub-Sections (VSS). A TTD has two possible states: *free* and *occupied* with a safety invariant stating that if a train is located on a TTD, then the state of the TTD must be set to *occupied*. In addition to these two states, a VSS may have the *unknown* or the *ambiguous* state. The *ambiguous* state is used when the information available to the system suggest that two trains are potentially present on the VSS. The *unknown* state is used when the system can guarantee neither the presence nor the absence of a train on the VSS. For an optimal safety, Movement Authorities (MA) are evaluated and assigned to each connected train. The MA of a train designates a portion of the track on which it is guaranteed to move safely. ERTMS (European Rail Traffic Management System) designates a protocol and a set of tools that allow a train to know and report its position. Similarly, TIMS (Train Integrity Monitoring System) designates the component that allows a train to know and report its integrity and its size. HEEL3 considers three train categories : those equipped with ERTMS and TIMS called *INTEGER*; those that are just equipped with a ERTMS which allows them to broadcast their position (connected trains); and finally, those that are equipped neither with a ERTMS nor with a TIMS called unconnected trains.

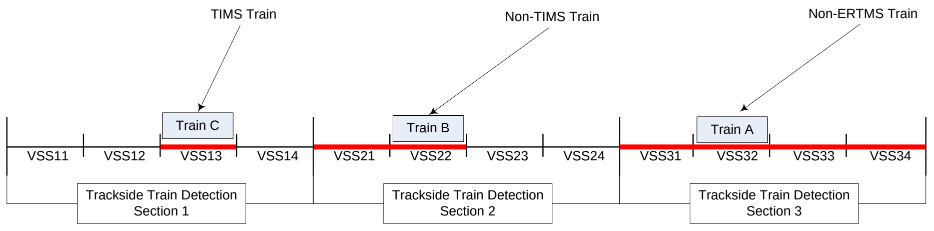

Fig. 1. Overview of the dependence between the capacity exploitation and the presence of ERTMS and TIMS [12]

Figure 1 is an overview of the influence of the presence of ERTMS and TIMS on the track capacity exploitation [12]. A TIMS train (INTEGER) is considered to occupy a whole VSS. A non-TIMS train (connected train) is considered to occupy all the VSSs from its front to the end of the TTD section where it is located. Finally, a non-ERTMS train (unconnected train) is considered to occupy the whole TTD section where the system guess it is.

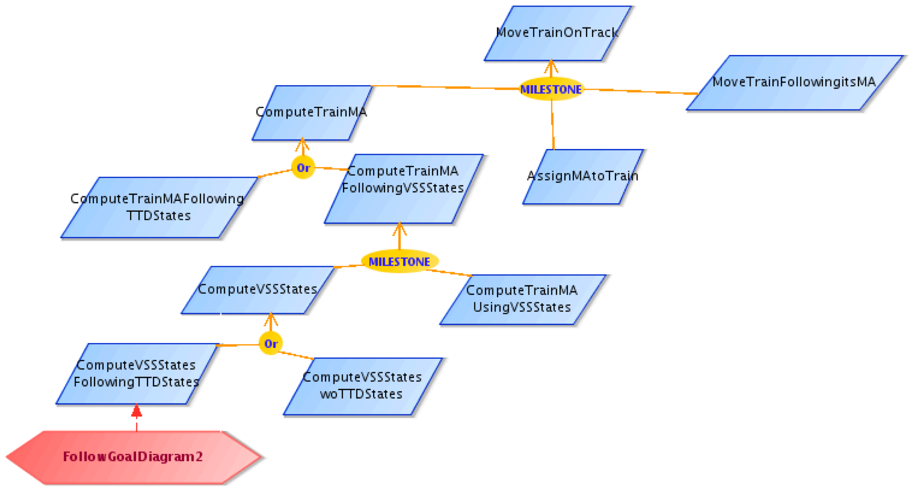
MoveTrainOnTrack
MILESTONE
MoveTrainFollowingitsMA
ComputeTrainMA
AssignMAtoTrain
Or
ComputeTrainMAFollowing TTDStates
ComputeTrainMA FollowingVSSStates
MILESTONE
ComputeVSSStates
ComputeTrainMA UsingVSSStates
Or
ComputeVSSStates FollowingTTDStates
ComputeVSSStates woTTDStates
FollowGoalDiagram2

Fig. 2. The SysML/KAOS goal diagram

3.2 The Goal Diagram

The SysML/KAOS requirements engineering method allows the progressive construction of system requirements from refinements of stakeholder needs. Thus, even if the management of VSSs is the purpose of the case study, we need to put it into perspective with more abstract objectives that will explain what VSSs are useful for. Figure 2 is an excerpt from the SysML/KAOS functional goal diagram focused on the main system purpose : move trains on the track (`MoveTrainOnTrack`). To achieve it, the system must ensures that the train has a valid MA (`ComputeTrainMA`). If the MA has been recomputed, then the system must assign the new MA to the train (`AssignMAtoTrain`). Finally, the train has to move following its assigned MA (`MoveTrainFollowingItsMA`). The second refinement level of the SysML/KAOS goal diagram focuses on the informations needed to determine the MA of a train : the MA computation can be based only on TTD states (`ComputeTrainMAFollowingTTDStates`) or following VSS states (`ComputeTrainMAFollowingVSSStates`) [5]. When the computation is only based on TTD states, it corresponds to the *ERTMS/ETCS Level 2* protocol. When VSS states are involved, it corresponds to the *ERTMS/ETCS Level 3* protocol. The MA computation based on VSS states requires the update of the states of VSSs (`ComputeVSSStates`) and the computation of the MA (`ComputeTrainMAUsingVSSStates`). Finally, depending on the type of the ERTMS/ETCS level 3 implementation, it is possible to use or not the TTD states when computing the VSS states (table 1 of [12]). If TTD states are not required (*virtual (without train detection)* level 3 type), it corresponds to `ComputeVSSStateswoTTDStates`, with the disadvantage of only allowing the circulation of trains equipped with TIMS. If TTD states are used (*hybrid* level 3 type), it corresponds to `ComputeVSSStatesFollowingTTDStates`.

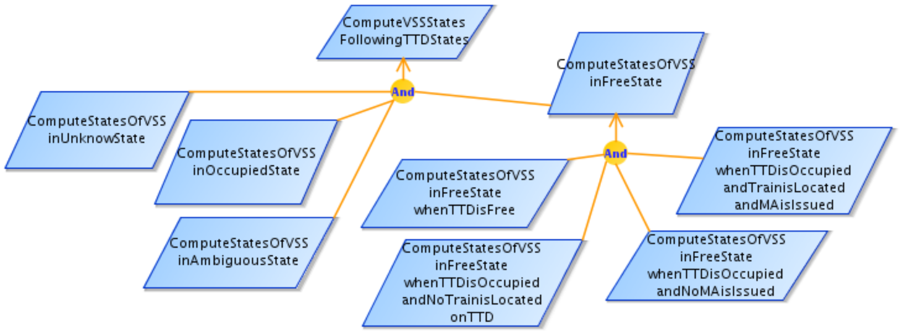
ComputeVSSStates FollowingTTDStates
ComputeStatesOfVSS inFreeState
And
ComputeStatesOfVSS inUnknowState
ComputeStatesOfVSS inOccupiedState
ComputeStatesOfVSS inAmbiguousState
And
ComputeStatesOfVSS inFreeState whenTTDisFree
ComputeStatesOfVSS inFreeState whenTTDisOccupied andTrainisLocated andMAisIssued
ComputeStatesOfVSS inFreeState whenTTDisOccupied andNoTrainisLocated onTTD
ComputeStatesOfVSS inFreeState whenTTDisOccupied andNoMAisIssued

Fig. 3. SysML/KAOS goal diagram of the VSS state computation purposes

Figure 3 is an excerpt from the SysML/KAOS functional goal diagram focused on the purpose of VSS state computation with the use of TTD states (`ComputeVSSStatesFollowingTTDStates`). The computation of the current VSS states can be splitted into the determination of the current states of VSSs previously in the unknown state (`ComputeStatesOfVSSinUnknownState`), in the occupied state (`ComputeStatesOfVSSinOccupiedState`), in the ambiguous state (`ComputeStatesOfVSSinAmbiguousState`) and in the free state (`ComputeStates-OfVSSinFreeState`) (Figure 7 of [5]). The last refinement level is focused on VSSs previously in the free state. Its goals come from the requirements of the transition *#1A* of Table 2 of [5]. When the TTD is free, then the VSSs remain free (`ComputeStatesOfVSSinFreeStateWhenTTDisFree`). When the TTD is occupied and no train is located on it or no MA is issued, then the VSSs move in the unknown state (`ComputeStatesOfVSSinFreeStateWhenTTDisOccupiedandNoTrain-isLocatedonTTD`, `ComputeStatesOfVSSinFreeStateWhenTTDisOccupiedandNoMA-isIssued`). The other transitions are the purpose of `ComputeStatesOfVSSinFree-StateWhenTTDisOccupiedandTrainisLocatedandMAisIssued`.

The rest of this section consists of a presentation of the SysML/KAOS domain models associated with the most relevant refinement levels of the goal diagrams and of a description of the *B System* specifications obtained from goals and ontologies. From the goal model, we distinguish seven refinement levels which are translated into seven *B System* components. The formal specification has been verified using *Rodin* [3], an industrial-strength tool supporting the *Event-B* method [1]. We have in particular discharged all the proof obligations associated with the safety invariants that we have identified and with the SysML/KAOS refinement operators that appear in the goal diagram. For the sake of concision, we will present here only the first three refinement levels. The full specification can be found in [19].

3.3 The Root Level

```
domain model ertms_etcs_case_study {
       concepts:
              concept TRAIN is variable: false
       attributes:
              attribute connectedTrain domain: Train range: BOOL {
                     is variable: true  is functional: true   is total: false
              }
              attribute front domain: dom(connectedTrain) range: TRACK {
                     is variable: true  is functional: true   is total: true
              }
              attribute rear domain: dom(connectedTrain) range: TRACK {
                     is variable: true  is functional: true   is total: false
              }
       data sets:
              custom data set TRACK
       data values:
              data value a type: NATURAL
              data value b type: NATURAL
       predicates:
              p0.1: a<b                          p0.2: TRACK=a..b
              p0.3: !tr. (tr : dom(rear) => rear(tr) < front(tr))
}
```

Fig. 4. SysML/KAOS domain modeling of the goal diagram root level

Figure 4 represents the domain model associated with the root level of the SysML/KAOS goal diagram of Figure 2. The concept `TRAIN` models the set of trains. The attribute `connectedTrain` models the subset of `TRAIN` that broadcast their location at least once and for each, the current connection status. The attribute `front` models the estimated position of the front of each connected train. For each connected train equipped with a TIMS, the attribute `rear` models the estimated position of its rear[6]. Thus, $dom(front) \setminus dom(rear)$ represents the set of trains equipped with a ERTMS and not equipped with a TIMS. Predicates represent constraints on domain model elements. Each predicate is prefixed with an identifier. For example, the predicate `p0.2` defines `TRACK` as the data range $a..b$.

Figure 5 represents the *B System* model obtained from the translation of the root level of the goal diagram of Figure 2 and of the associated domain model of Figure 4. The domain model gives rise to sets, constants, properties, variables and invariants of the formal specification. Predicates involving variables give rise to invariants and the others to properties. The *isFunctional* and *isTotal* characteristics of attributes, are used to guess if an attribute should be translated into a partial or total function. The root goal is translated into an event for which the body has been manually specified: the movement of a connected train (`grd1`) results in the incrementation of the position of its front (`act1`) and its rear (`act2` in the case of an *INTEGER* train) of the value corresponding to the movement. Of course, the movement can only be done if the train stays on the track (`grd3`).

[6] the rear is deduced from the front and length of the train, since a train equipped with a TIMS broadcast its length and its integrity

SYSTEM ertms_etcs_case_study
SETS TRAIN
CONSTANTS a b TRACK
PROPERTIES
  axm1: $a \in \mathbb{N}$ axm2: $b \in \mathbb{N}$ p0.1: $a < b$
  p0.2: $TRACK = a \mathinner{..} b$
VARIABLES connectedTrain front rear
INVARIANT
  inv1: $connectedTrain \in TRAIN \mathrel{+\!\!\!\!\rightarrow} BOOL$
  inv2: $front \in$
    $dom(connectedTrain) \rightarrow TRACK$
  inv3: $rear \in$
    $dom(connectedTrain) \mathrel{+\!\!\!\!\rightarrow} TRACK$
  p0.3: $\forall tr \cdot (tr \in dom(rear)$
    $\Rightarrow rear(tr) < front(tr))$
Event MoveTrainOnTrack $\widehat{=}$
  any tr len
  where
    grd1: $tr \in connectedTrain^{-1}[\{TRUE\}]$
    grd2: $len \in \mathbb{N}_1$
    grd3: $front(tr) + len \in TRACK$
  then
    act1: $front(tr) := front(tr) + len$
    act2: $rear := (\{TRUE \mapsto rear \mathbin{<\!\!\!+} \{tr \mapsto$
    $rear(tr) + len\}, FALSE \mapsto rear$
    $\})(bool(tr \in dom(rear)))$
  END END

Fig. 5. *B System* specification of the root level of the goal diagram of Figure 2

3.4 The First Refinement Level

```
domain model ertms_etcs_case_study_ref_1 parent domain model ertms_etcs_case_study {
        attributes:
                attribute MA domain: dom(connectedTrain) range: POW(TRACK) {
                        is variable: true  is functional: true   is total: false
                }
        predicates:
                p1.1: !tr. (tr : dom(MA) => #p,q.(p..q<:TRACK & p<=q & MA(tr)=p..q)))
                p1.2: !tr. (tr : dom(MA) => (front(tr) : MA(tr)))
                p1.3: !tr. (tr : dom(rear) & tr : dom(MA) => rear(tr) : MA(tr))
                p1.4: !tr1,tr2. ((tr1 : dom(MA) & tr2 : dom(MA) & tr1 /=
                        tr2)=>MA(tr1) /\ MA(tr2)={})
}
```

Fig. 6. SysML/KAOS domain modeling of the goal diagram first refinement level

Figure 6 represents the domain model associated with the first refinement level of the SysML/KAOS goal diagram of Figure 2. It refines the one associated with the root level and introduces an attribute named `MA` representing the MA assigned to a connected train. The MA of a train is modeled as a contiguous part of the track (`p1.1`), containing the train (`p1.2` and `p1.3`). Finally, the predicate `p1.4` asserts that the MA assigned to two different trains must be disjoint. The predicates `p1.2` and `p1.3` are gluing invariants, linking the concrete variable `MA` with the abstract variables `front` and `rear`.

Figure 7 represents the *B System* model obtained from the translation of the first refinement level of the goal diagram of Figure 2 and of the associated domain model of Figure 6. Each refinement level goal is translated into an event for which the body has been manually specified : the current MA of the train is computed and stored into a variable named `MAtemp` (event `ComputeTrainMA`). Because the computation of the MA is out of the scope of the case study [7], the event simply nondeterministically choose an MA, with respect to the safety invariants. This MA is then assigned to the train by updating the variable `MA` (event `AssignMAtoTrain`) and taken into account for the train displacement (event `MoveTrainFollowingItsMA`). Theorems `s1`, `s2`, `s3` and `s4` represent the proof obligations related to the usage of the *MILESTONE* operator between the root and the first refinement levels. Since each proof obligation has been

```
REFINEMENT ertms_etcs_case_study_ref_1
REFINES ertms_etcs_case_study
VARIABLES connectedTrain front rear MA MAtemp
INVARIANT
   inv1: $MA \in dom(connectedTrain) ⇸ \mathbb{P}(TRACK)$
   p1.1: $\forall tr \cdot (tr \in dom(MA) \Rightarrow (\exists p, q \cdot (p\,..\,q \subseteq TRACK \land p \leq q \land MA(tr) = p\,..\,q)))$
   p1.2: $\forall tr \cdot (tr \in dom(MA) \Rightarrow front(tr) \in MA(tr))$
   p1.3: $\forall tr \cdot (tr \in dom(rear) \cap dom(MA) \Rightarrow rear(tr) \in MA(tr))$
   p1.4: $\forall tr1, tr2 \cdot ((\{tr1, tr2\} \subseteq dom(MA) \land tr1 \neq tr2) \Rightarrow MA(tr1) \cap MA(tr2) = \emptyset)$
   inv6: $MAtemp \in dom(connectedTrain) ⇸ \mathbb{P}(TRACK)$
   inv7: $\forall tr \cdot (tr \in dom(MAtemp) \Rightarrow (\exists p, q \cdot (p\,..\,q \subseteq TRACK \land p \leq q \land MAtemp(tr) = p\,..\,q)))$
   theorem s1: $ComputeTrainMA\_Guard \Rightarrow MoveTrainOnTrack\_Guard$
   theorem s2: $ComputeTrainMA\_Post \Rightarrow AssignMAtoTrain\_Guard$
   theorem s3: $AssignMAtoTrain\_Post \Rightarrow MoveTrainFollowingItsMA\_Guard$
   theorem s4: $MoveTrainFollowingItsMA\_Post \Rightarrow MoveTrainOnTrack\_Post$
Event
 ComputeTrainMA ≙
   any tr p q len
   where
     grd1: $tr \in connectedTrain^{-1}[\{TRUE\}]$
     grd2: $p\,..\,q \subseteq TRACK \land p \leq q$
     grd3: $front(tr) \in p\,..\,q$
     grd4: $tr \in dom(rear) \Rightarrow rear(tr) \in p\,..\,q$
     grd5: $p\,..\,q \cap union(ran(\{tr\} ⩤ MA)) = \emptyset$
     grd6: $len \in \mathbb{N}_1$
     grd7: $front(tr) + len \in TRACK$
   then
     act1: $MAtemp(tr) := p\,..\,q$
   END

 AssignMAtoTrain ≙
   any tr len
   where
     grd1: $tr \in connectedTrain^{-1}[\{TRUE\}] \cap dom(MAtemp)$
     • • •
     grd6: $front(tr) + len \in MAtemp(tr)$
   then
     act1: $MA(tr) := MAtemp(tr)$
   END

 MoveTrainFollowingItsMA ≙
   any tr len
   where
     grd1: $tr \in connectedTrain^{-1}[\{TRUE\}] \cap dom(MA)$
     grd2: $len \in \mathbb{N}_1$
     grd3: $front(tr) + len \in MA(tr)$
   then
     act1: $front(tr) := front(tr) + len$
     act2: $rear := (\{TRUE \mapsto rear \mathbin{<\!\!+} \{tr \mapsto rear(tr) + len\}, FALSE \mapsto rear\})(bool(tr \in dom(rear)))$
   END
 END
```

Fig. 7. *B System* specification of the first refinement level of the diagram of Figure 2

modeled as an *Event-B* theorem, it has been proved based on system properties and invariants. To deal with the fact that *Event-B* does not currently support the temporal logic, we have used the proof obligation $G1\_Post \Rightarrow G2\_Guard$ for the invariants `s2` and `s3`, instead of $\Box(G1\_Post \Rightarrow \Diamond G2\_Guard)$ (Sect. 2.3), since $(G1\_Post \Rightarrow G2\_Guard) \Rightarrow (\Box(G1\_Post \Rightarrow \Diamond G2\_Guard))$. The full specification of `s1` is given below:

theorem s1: $\forall tr, p, q, len \cdot (((tr \in connectedTrain^{-1}[\{TRUE\}]) \land (p\,..\,q \subseteq TRACK \land p \leq q) \land (front(tr) \in p\,..\,q) \land (tr \in dom(rear) \Rightarrow rear(tr) \in p\,..\,q) \land (p\,..\,q \cap union(ran(\{tr\} ⩤ MA)) = \emptyset) \land (len \in \mathbb{N}_1) \land (front(tr) + len \in TRACK)) \Rightarrow ((tr \in connectedTrain^{-1}[\{TRUE\}]) \land (len \in \mathbb{N}_1) \land (front(tr) + len \in TRACK)))$

It expresses the fact that the activation of the guard of `ComputeTrainMA` for certain parameters is sufficient for the activation of the guard of `MoveTrainOnTrack` for this same group of parameters.

3.5 The Second Refinement Level

Figure 8 represents the domain model associated with the second refinement level of the diagram of Figure 2. It refines the one associated with the first refinement level and introduces two concepts named `TTD` and `VSS`. The attributes `stateTTD` and `stateVSS` represent the states of the corresponding con-

```
domain model ertms_etcs_case_study_ref_2 parent domain model ertms_etcs_case_study_ref_1 {
       concepts:
              concept TTD is variable: false      concept VSS is variable: false
       attributes:
              attribute stateTTD domain: TTD range: TTD_STATES {
                     is variable: true  is functional: true   is total: true
              }
              attribute stateVSS domain: VSS range: VSS_STATES {
                     is variable: true  is functional: true   is total: true
              }
       data sets:
              enumerated data set VSS_STATES {
                     elements :
                             data value OCCUPIED   data value FREE
                             data value UNKNOWN   data value AMBIGUOUS
              }
              enumerated data set TTD_STATES {
                     elements :  data value OCCUPIED    data value FREE
              }
       predicates:
              p2.1: TTD <: POW1(TRACK)
              p2.2: union(TTD) = TRACK
              p2.3: inter(TTD) = {}
              p2.4: !ttd. (ttd : TTD => #p,q.(p..q<:TRACK & p<q & ttd=p..q)))
              p2.5: VSS <: POW1(TRACK)
              p2.6: union(VSS) = TRACK
              p2.7: inter(VSS) = {}
              p2.8: !vss. (vss : VSS => #p,q,ttd.(ttd : TTD & p..q<:ttd & p<q & vss=p..q)))
              p2.9: !ttd,tr. ( tr : dom(front) \ dom(rear) & ttd : TTD & front(tr) : ttd)
                      => (( ttd |-> OCCUPIED ) : stateTTD)
              p2.10: !ttd,tr. (tr : dom(rear) & ttd : TTD & (rear(tr)..front(tr))/\ttd /= {})
                     => (( ttd |-> OCCUPIED ) : stateTTD)
              p2.11: !tr1,tr2. (tr1 : dom(rear) & tr2 : dom(rear) & tr1 /= tr2)
                     => ( (rear(tr1)..front(tr1))/\(rear(tr2)..front(tr2))={})
              p2.12: !tr1,tr2,ttd.(tr1 : dom(rear) & tr2 : dom(front)\dom(rear) & tr1 /= tr2
                            & ttd : TTD & front(tr2) : ttd & rear(tr1)..front(tr1))/\ttd /= {})
                            => ( front(tr2)<rear(tr1))
              p2.13: !tr1,tr2,ttd. ( tr1 : dom(front)\dom(rear) & tr2 : dom(front)\dom(rear)
                     &tr1 /= tr2 & ttd : TTD & front(tr1) : ttd) => ( front(tr2) /: ttd)
}
```

Fig. 8. SysML/KAOS domain modeling of the goal diagram second refinement level

cepts. The predicates `p2.1`..`p2.8` define each TTD as a contiguous part of the track and each VSS as a contiguous part of a TTD. The predicates `p2.9` and `p2.10` are used to state that if a train is located on a TTD, then its state must be occupied: a train $tr \in TRAIN$ is located on $ttd \in TTD$ if $front(tr) \in ttd$ (`p2.9`) or if tr is equipped with a TIMS ($tr \in dom(rear)$) and $(rear(tr)..front(tr)) \cap ttd \neq \emptyset$ (`p2.10`). Finally, the predicates `p2.11`..`p2.13` states that two different trains must be in disjoint parts of the track: for two trains `tr1` and `tr2`, if they are equipped with TIMS, then the track portions that they occupy should just be disjointed (`p2.11`); if they are on the same TTD and one of them, (`tr2`), is not equipped with a TIMS, then, the second, (`tr1`), must be equipped with a TIMS and `tr2` must be in the rear of `tr1` (`p2.12`); if none of them is an INTEGER train, then they must be in two distincts TTDs (`p2.13`). The predicates `p2.9` and `p2.10` are gluing invariants, linking the concrete variable `stateTTD` with the abstract variables `front` and `rear`.

The *B System* specification raised from the translation of the second refinement level includes the result of the translation of the domain model of Figure 8, two new events (`ComputeTrainMAFollowingTTDStates`, `ComputeTrainMAFollowingVSSStates`), an extension of the event `MoveTrainFollowingItsMA` taking into account the new safety invariants and the theorems representing the proof obligations related to the usage of the *OR* operator between the first and second refinement levels.

4 Discussion

Benefits This case study allowed us to benefit from the advantages of a high-level modeling approach within the framework of the formal specification of the hybrid ERTMS/ETCS level 3 requirements : decoupling between formal specification handling difficulties and system modeling; better reusability and readability of models; strong traceability between the system formal specification and the goal model, which is an abstraction of the case study description. Using the FORMOSE approach, we have quickly build the refinement hierarchy of the system and we have determined and formally expressed the safety invariants. The approach bridges the gap between the system textual description and its formal specification. Its use has made it possible to better present the specifications, excluding predicates, to stakeholders and to better delineate the system boundaries. Using Rodin [3], we have formally verified and validated the safety invariants and the goal diagram refinement hierarchy. Through proB, we have animated the formal model. The full specification can be found in [19]. One conclusion of our work is that the description of the standard, as it exists in the documents [7,5,12], does not guarantee the absence of train collisions. Indeed, since the standard allows the movement of unconnected trains on the track, nothing is specified to guarantee that an unconnected train will not hit another train (connected or not). The animation of the specification allows the observation of these states. The only guarantee that the safety invariants expressed in [7,5,12] bring is that a connected train will never hit another train.

Comparison We have also specified in a companion paper [9] the case study using plain Event-B, in the traditional style. Two distinct specifiers (first author of [9] and first author of this paper) wrote each specification without interacting with each other during specification construction. Critical reviewing by the team was then conducted after the specifications were built. The specification in [9] includes four refinement levels. The TTDs and trains are introduced in the root level and the VSSs are introduced in the second refinement level, as refinements of TTDs. The MAs and VSS states are introduced in the third refinement level (M3), for train movement supervision. A strategy is proposed to prove the determinism of the transitions of VSS states. The state variables of [9] are partitioned into environment variables and controller variables, and similarly for events. Environment events only modify environment state variables. Controller events read environment variables and update controller variables. In this paper, we only model controller events; state variables represent the controller view of the environment. The execution ordering and the refinement strategy are enforced using proof obligations expressed as theorems, whereas in [9] there is no proof about these aspects. In [9], the safety properties are introduced in the last refinement level; here, we introduce them in the first (predicate `p1.4`) and second (predicates `p2.9`..`p2.13`) refinements. In [9], all trains equipped with ERTMS are equipped with TIMS, so they broadcast their front and rear; here, we consider ERTMS trains with or without TIMS, so a ERTMS train may or may not broadcast its rear. The FORMOSE approach makes it possible to trace the source and justify the need for each formal component and its contents, in relation with the SysML/KAOS goal and domain models.

Difficulties The expression of theorems representing proof obligations associated to SysML/KAOS refinement operators was difficult because there is no way in Rodin to designate the guard and the post condition of an event within predicates. Table 2 summarises the key characteristics related to the formal specification. It seemed that the provers have a lot of trouble with data ranges such as $p..q$ and with conditional actions such as $rear := (\{TRUE \mapsto rear \lhd\!\!- \{tr \mapsto rear(tr) + len\}, FALSE \mapsto rear\})(bool(tr \in dom(rear)))$ defined in the component `ertms_etcs_case_study` to simulate an *if-then-else* in order to avoid the definition of a second event.

Table 2. Key characteristics related to the formal specification

Refinement level	L0	L1	L2	L3	L4	L5	L6
Invariants	4	11	13	4	6	5	9
Proof Obligations (PO)	20	40	50	13	5	5	14
Automatically Discharged POs	17	30	30	11	0	0	4
Interactively Discharged POs	3	5	20	2	5	5	10

5 Conclusion and Future Work

This paper focusses on the use of the FORMOSE approach for the high level modeling of system requirements, of domain properties and of safety invariants related to the hybrid ERTMS/ETCS level 3 standard [7,5,12]. Translation rules, supported by tools [11,19], have then been applied to obtain a formal specification containing the system structure and the skeleton of events. The Rodin tool [3] has been used to verify and validate the formal specification, especially to prove the safety invariants and the refinement logic, after the completion of the body of events. The full specification can be found in [19]. A comparison with a companion paper on the same case study, but specified using only plain *Event-B*, has been done.

Work in progress aims at improving the representation of domain predicates (to make them more user-friendly) and at evaluating the impact of updates on *B System* specifications within SysML/KAOS models. We are also working on integrating the approach within the open-source platform *Openflexo* [13] which federates the various contributions of *FORMOSE* project partners [2].

Acknowledgment

This work is carried out within the framework of the *FORMOSE* project [2] funded by the French National Research Agency (ANR). It is also partly supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

1. Abrial, J.: Modeling in Event-B - System and Software Engineering. Cambridge University Press (2010)
2. ANR-14-CE28-0009: Formose ANR project (2017)
3. Butler, M.J., Jones, C.B., Romanovsky, A., Troubitsyna, E. (eds.): Rigorous Development of Complex Fault-Tolerant Systems, LNCS, vol. 4157. Springer (2006)
4. ClearSy: Atelier B: B System (2014), `http://clearsy.com/`
5. EEIG ERTMS Users Group: Hybrid ERTMS/ETCS Level 3: Principles. Ref. 16E042 Version 1A (Jul 2017)
6. Gnaho, C., Semmak, F., Laleau, R.: Modeling the impact of non-functional requirements on functional requirements. LNCS, vol. 8697, pp. 59–67. Springer (2013)
7. Hoang, T.S., Butler, M., Reichl, K.: The Hybrid ERTMS/ETCS Level 3 Case Study. ABZ pp. 1–3 (2018)
8. van Lamsweerde, A.: Requirements Engineering - From System Goals to UML Models to Software Specifications. Wiley (2009)
9. Mammar, A., Frappier, M., Tueno, S., Laleau, R.: An event-b model of the hybrid ertms/etcs level 3 standard (2018), `info.usherbrooke.ca/mfrappier/abz2018-ERTMS-Case-Study`
10. Mammar, A., Laleau, R.: On the use of domain and system knowledge modeling in goal-based Event-B specifications. In: ISoLA 2016, LNCS. pp. 325–339. Springer
11. Matoussi, A., Gervais, F., Laleau, R.: A goal-based approach to guide the design of an abstract Event-B specification. In: ICECCS 2011. pp. 139–148. ICS
12. Nicola, F., Henri, v.H., Laura, A., Maarten, B.: ERTMS Level 3: the Game-Changer. IRSE News View p. 232 (Apr 2017)
13. Openflexo: Openflexo project (2015), `http://www.openflexo.org`
14. Pierra, G.: The PLIB ontology-based approach to data integration. In: IFIP 18th World Computer Congress. IFIP, vol. 156, pp. 13–18. Kluwer/Springer (2004)
15. Sengupta, K., Hitzler, P.: Web ontology language (OWL). In: Encyclopedia of Social Network Analysis and Mining, pp. 2374–2378 (2014)
16. Tueno, S., Laleau, R., Mammar, A., Frappier, M.: Towards Using Ontologies for Domain Modeling within the SysML/KAOS Approach. IEEE proceedings of MoDRE workshop, 25th IEEE International Requirements Engineering Conference
17. Tueno, S., Laleau, R., Mammar, A., Frappier, M.: Formal Representation of SysML/KAOS Domain Models (Complete Version). ArXiv e-prints, cs.SE, 1712.07406 (Dec 2017)
18. Tueno, S., Laleau, R., Mammar, A., Frappier, M.: The SysML/KAOS Domain Modeling Approach. ArXiv e-prints, cs.SE, 1710.00903 (Sep 2017)
19. Tueno, S., Laleau, R., Mammar, A., Frappier, M.: The SysML/KAOS Domain Modeling Language (Tool and Case Studies) (2017), `https://github.com/stuenofotso/SysML_KAOS_Domain_Model_Parser/tree/master`