Modeling the Hybrid ERTMS/ETCS Level 3 Implementation through Goal Diagrams and Ontologies Using the FORMOSE 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 implementation in the framework of the case study proposed for the 6th edition of the ABZ conference. The specification is based on the methodology and tools, raised from the *ANR FORMOSE* project, for the modeling and formal validation of critical and complex system requirements. The requirements are captured as *SysML/KAOS* goal diagrams and are automatically translated into *B System* specifications, in order to obtain the backbone of the formal specification. Domain properties are captured as ontologies through the SysML/ KAOS domain modeling language, based on *OWL* and *PLIB*. From these ontologies is automatically extracted the structural part of the system formal specification that completes the result of the translation of goal diagrams. The system construction is thus incremental, based on refinement mechanisms existing within the involved methods and leads to a formally correct system, while eliminating any unnecessary manipulation of the formal specification. These manipulations being often very difficult and sources of many failures. The only part of the formal specification, which must be manually completed, is the body of events.

Keywords: Requirement Engineering, Goal Diagrams, Domain Modeling, Ontologies, *SysML/KAOS*, *B System*, *Event-B*

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 [8]. This case study deals with the specification of the *hybrid ERTMS/ETCS level 3* implementation [5, 14]. The case study is described through two main documents. The first, [8], describes the hybrid ERTMS/ETCS level 3 protocol in a general way and restricts the scope of the study. The second, [5], offers a technical and detailed description of the protocol specification. It provides the safety requirements that the system must guarantee. The FORMOSE method consists in the usage of the *SysML/KAOS* requirements engineering method [7, 11] for capturing system requirements as goal diagrams. Domain properties are modeled as ontologies, using the SysML/KAOS domain modeling language [22, 23]. Once done, translation rules [12, 18, 21], supported by tools [13, 20], allow the automatic generation of the system *B system* specification : the requirements leading to the framework of the formal model and the ontologies to the system structure. The framework consists of components that constitute the formal model as well as refinement links between them. Each component containing the skeleton of events and the proof obligations characterising the decomposition hierarchy within SysML KAOS goal diagrams. The system structure 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 resulting from the application of the approach on the case study can be found in [19].

The remainder of this paper is structured as follows: Section 2 briefly describes the Event-B and B System formal methods, the SysML/KAOS requirements engineering method, the SysML/KAOS domain modeling language and the rules for obtaining the B System specifications from these high-level models. Follows a presentation, in Section 3, of the work done on the case study and in Section 4, of the discussion related to it. Finally, Section 5 reports our conclusions.

2 Context

2.1 Event-B and 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 useful properties. An Event-B model includes a static part called context and a dynamic part called machine. The context contains the declarations of abstract and enumerated sets, constants and axioms. The machine contains variables, invariants and events. Moreover, a machine can see contexts. 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. The system specification can be constructed using stepwise refinement. A machine can refine another machine, by adding new events or by reducing nondeterminacy of existing events. A refinement step can also introduce new state variables or replace abstract variables by more concrete ones. Furthermore, a context can be extended by another one. 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].

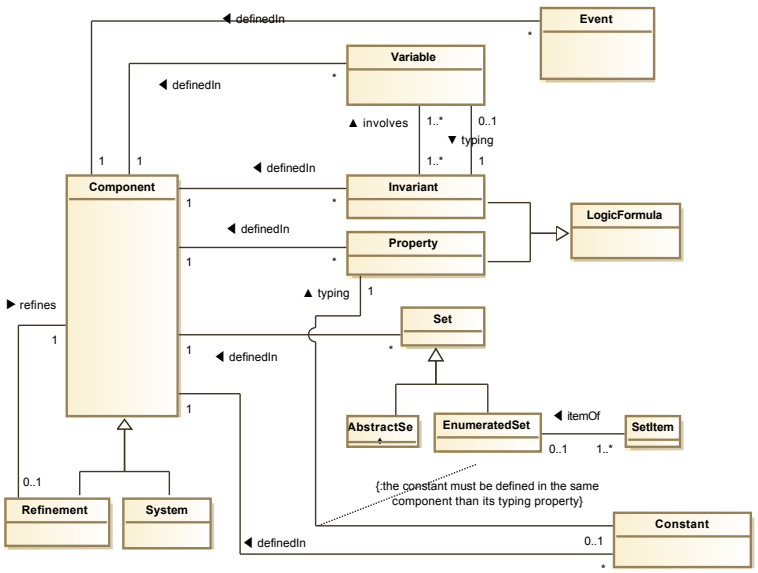

Fig. 1. Excerpt from a metamodel of the B System specification language

Figure 1 is an excerpt from a metamodel of the *B System* language. 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. A component defining static elements corresponds to an Event-B machine and the one defining dynamic elements corresponds to a context. 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. 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

Requirements engineering focuses on defining and handling requirements. These and all related activities, in order to be carried out, require the choice of an adequate means for requirements representation. The *KAOS* method [9] proposes to represent the requirements in the form of goals through five sub-models of which the two main ones are : the **goal model** for the representation of requirements to be satisfied by the

system and of expectations with regard to the environment through a hierarchy of goals and the **object model** which uses the *UML* class diagram for the representation of the domain vocabulary. The hierarchy is built through a succession of refinements using two main operators : ***AND*** and ***OR***. An ***AND*** *refinement* decomposes a goal into subgoals, and all of them must be achieved to realise the parent goal. Dually, 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. Requirements and expectations correspond to the lowest level goals of the model. However, *KAOS* offers no mechanism to maintain a strong traceability between requirements and design deliverables, making it difficult to validate them against the needs formulated.

SysML/KAOS [6, 11] comes in response to this solicitation by adding to *KAOS* the *SysML UML profile* specially designed by the Object Management Group (OMG) for the analysis and specification of complex systems. *SysML* allows for the capturing of requirements and the maintaining of traceability links between those requirements and design diagrams resulting from the system design phase. Despite these advantages, OMG has not defined a precise syntax for requirements specification. SysML/KAOS therefore proposes to extend the SysML metamodel with a set of concepts allowing to represent requirements in SysML models using the KAOS expressivity.

SysML/KAOS goals can be *functional* or *non-functional*. The scope of this document is limited to functional requirements. A functional goal, under SysML/KAOS, describes the *expected behaviour* of the system once a certain condition holds [11] : *[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 [12]. The proposed rules allow the generation of a formal model whose structure reflects the hierarchy of the SysML/KAOS goal diagram : one machine is associated with each hierarchy level; this machine defines one event for each goal. The refinement links between goals are materialized within the formal specification through refinement links between machines and through a set of proof obligations which complement proof obligations for *invariant preservation* and for *event actions feasibility*; the other usual proof obligations being irrelevant for our purposes. Regarding the new proof obligations, they depend on the refinement operator used. In the case of the *AND* operator, it is necessary to demonstrate that the guard of each concrete event strengthens the guard of the abstract event and that the interleaving of concrete events actions changes the state of the system in a way that does not contradict the action of the abstract event. For the *OR* operator, it is also necessary to demonstrate that the guard of each concrete event strengthens the guard of the abstract event. Furthermore, it is necessary to prove that the action of each concrete event changes the state of the system in a way that does not contradict the action of the abstract event. Finally, the *OR* operator requires that one and only one of the concrete events may be triggered, which means that the execution of one of the concrete events must prevent the possible execution of another one. The *MILESTONE* operator requires the demonstration of the *scheduling constraint* : the end of an event is sufficient for the triggering of the next event in the sequence. It also requires that the guard of the first event in the sequence strengthens the guard of the abstract event and that the action of the last event in the sequence changes the state of the system in a way that does not contradict the action of the abstract event.

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

2.4 SysML/KAOS Domain Modeling

Domain models in SysML/KAOS are represented using ontologies. These ontologies are expressed using the SysML/KAOS domain modeling language [22, 23], a language based on *OWL* [17] and *PLIB* [16], two well-known ontology modeling languages.

Figure 2 is an excerpt from the metamodel associated with the SysML/KAOS domain modeling language. The *parent* association represents the hierarchy of domain models. Each domain model corresponds to a refinement level in the SysML/KAOS goal model. A *concept* (instance of Concept) represents a group of individuals sharing common characteristics. A *concept* can be declared *variable* (*isVariable=TRUE*) when

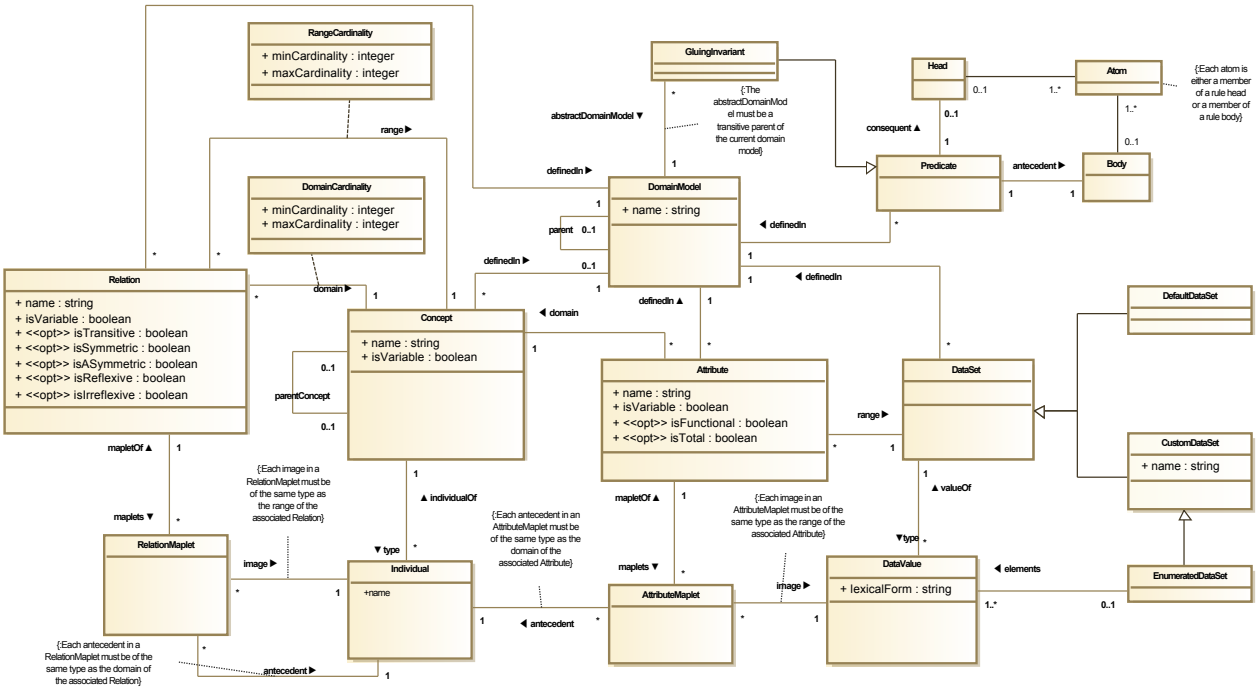

Fig. 2. Metamodel associated with the SysML/KAOS domain modeling language

the set of its individuals is likely to be updated through addition or deletion of individuals. Otherwise, it is considered to be *constant* (*isVariable=FALSE*). Instances of Relation are used to capture links between concepts, and instances of Attribute capture links between concepts and data sets. The most basic way to build an instance of DataSet is by listing its elements. This can be done through the DataSet specialization called EnumeratedDataSet. A relation or an attribute can be declared *variable* if the list of maplets related to it is likely to change over time. Otherwise, it is considered to be *constant*. Instances of RelationMaplet are used to define associations between instances of Individual through instances of Relation.

The notion of Predicate is used to represent constraints between different elements of the domain model in the form of *Horn clauses*: each predicate has a body which represents its *antecedent* and a head which represents its *consequent*, body and head designating conjunctions of atoms.

2.5 From SysML/KAOS Domain Models to B System Specifications

The translation rules are fully described in [18, 21]. They allow the extraction of the structural part of the system formal specification from domain model ontologies in order to complete the result of the formalisation of SysML/KAOS domain models. Table 1 represents the main rules. 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	
Translation Of	**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
Concept with parent	CO PCO	$\{CO, PCO\} \subseteq$ Concept CO is associated to PCO through the *parentConcept* association PCO has already been translated	o_CO	$o\_CO \in$ Constant $o\_CO \subseteq o\_PCO$

Relation	RE CO1 CO2	$\{CO1, CO2\} \subseteq$ Concept $RE \in$ Relation $CO1$ is the *domain* of RE $CO2$ is the *range* of RE $CO1$ and $CO2$ have already been translated	o_RE	**IF** the $isVariable$ property of RE is set to $FALSE$ **THEN** $o\_RE \in$ Constant **ELSE** $o\_RE \in$ Variable **END** $o\_RE \in o\_CO1 \leftrightarrow o\_CO2$
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**
Concept variability	CO	$CO \in$ Concept the $isVariable$ property of CO is set to $TRUE$ CO has already been translated	X_CO	$X\_CO \in$ Variable $X\_CO \subseteq o\_CO$
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 $o\_Dva \in o\_DS$
Relation maplets	RE $(M_j)_{j=1..n}$ $(a_j, i_j)_{j=1..n}$	$RE \in$ Relation $(M_j)_{j=1..n}$ are *maplets* of RE $\forall j \in 1..n, a_j$ is the antecedent of M_j $\forall j \in 1..n, i_j$ is the image of M_j RE and $(a_j, i_j)_{j=1..n}$ have already been translated		**IF** the $isVariable$ property of RE is set to $FALSE$ **THEN** $o\_RE = \{(o\_a_j, o\_i_j)_{j=1..n}\}$ (Property) **ELSE** $o\_RE := \{(o\_a_j, o\_i_j)_{j=1..n}\}$ (Initialisation) **END**

3 Formose Specification of a Hybrid ERTMS/ETCS level 3 System

The Hybrid ERTMS/ETCS level 3 protocol (HEEL3) has been proposed to optimize the use and occupation of railways [8, 5, 14]. 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.

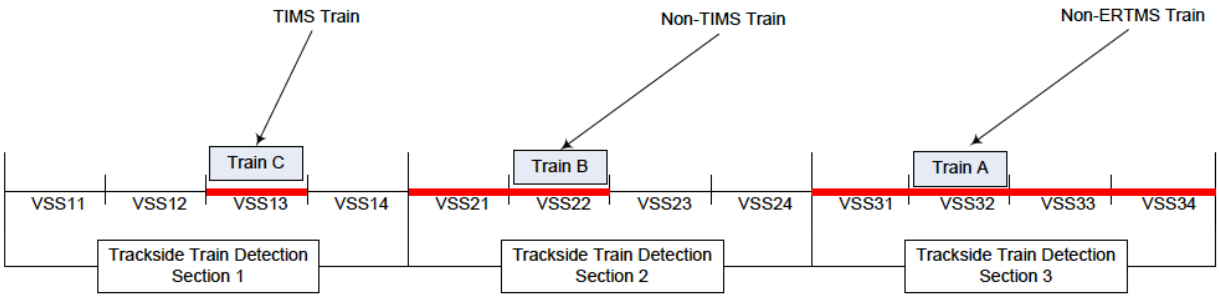

Fig. 3. Overview of the dependence between the capacity exploitation and the presence of ERTMS and TIMS [14]

Figure 3 is an overview of the influence of the presence of ERTMS and TIMS on the track capacity exploitation [14]. A TIMS train (INTEGER) is considered to occupy a whole VSS. A non-TIMS train (con-

nected train) is considered to occupy all the VSS 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.

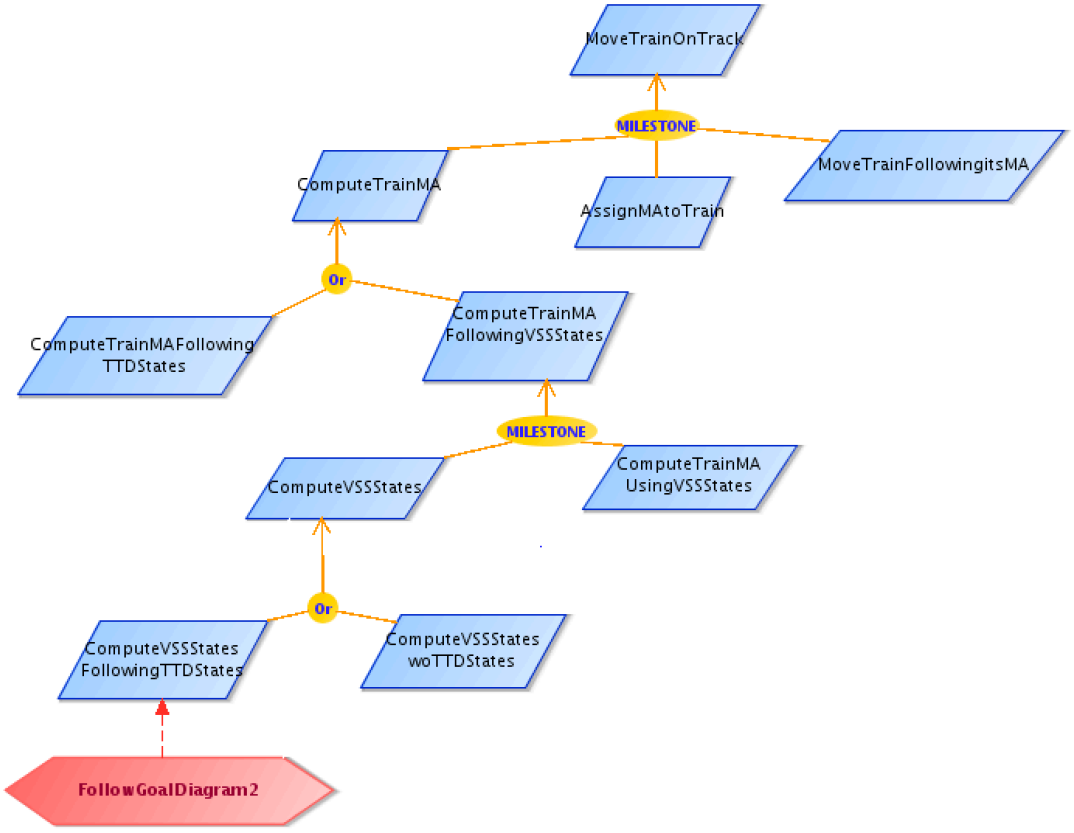

Fig. 4. SysML/KAOS goal diagram of the main system purposes

The SysML/KAOS requirements engineering method allows the progressive construction of system requirements from refinements of stakeholder needs. Figure 4 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 to be edited, then the system must computes the new MA and assigns it 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 VSS (`ComputeVSSStates`) followed by 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 [14]). If TTD states are not required, it corresponds to **Virtual (without train detection)** (`ComputeVSSStateswoTTDStates`), with the disadvantage of only allowing the circulation of trains equipped with TIMS. If TTD states are used, it corresponds to the hybrid ERTMS/ETCS level 3 implementation (`ComputeVSSStatesFollowingTTDStates`).

Figure 5 is an excerpt from the SysML/KAOS functional goal diagram focused on the purpose of VSS state computation with the usage of TTD states (`ComputeVSSStatesFollowingTTDStates`). The computation of

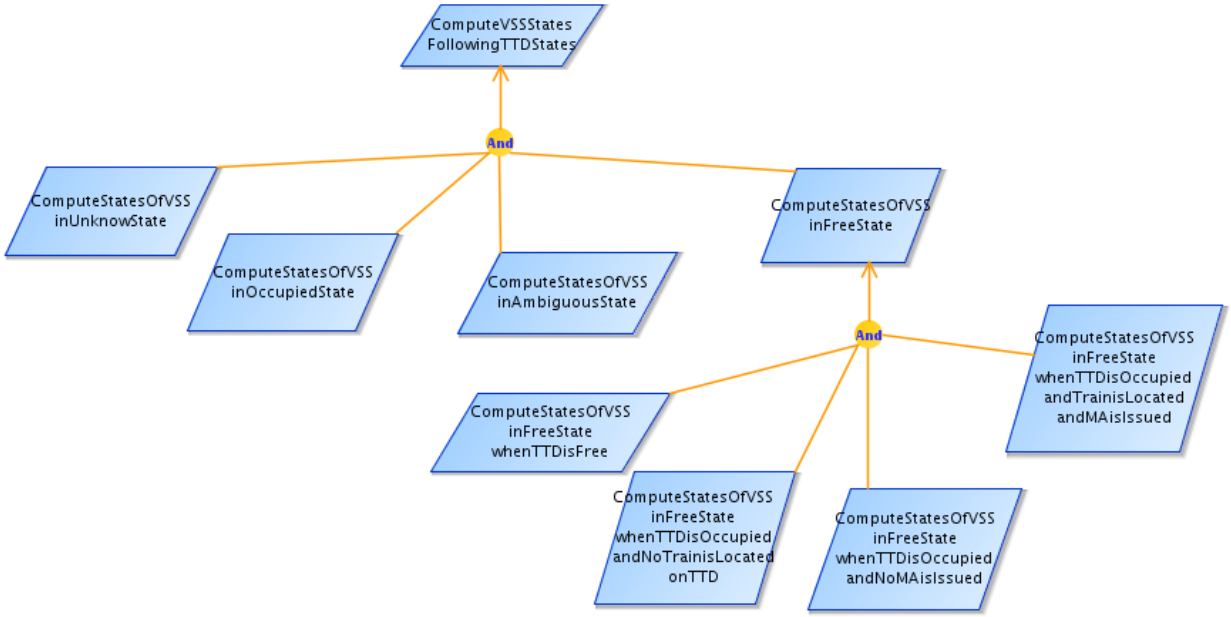

Fig. 5. SysML/KAOS goal diagram of the VSS state computation purposes

the current VSS states can be splitted into the determination of the current states of VSS previously in the unknown state (`ComputeStatesOfVSSinUnknownState`), in the occupied state (`ComputeStatesOfVSSinOccupiedState`), in the ambiguous state (`ComputeStatesOfVSSinAmbiguousState`) and in the free state (`ComputeStatesOfVSSinFreeState`) (Figure 7 of [5]). The last refinement level is focused on VSS 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 VSS remain free (`ComputeStatesOfVSSinFreeStateWhenTTDisFree`). When the TTD is occupied and no train is located on it or no MA is issued, then the VSS move in the unknown state (`ComputeStatesOfVSSinFreeStateWhenTTDisOccupiedandNoTrainisLocatedonTTD`, `ComputeStatesOfVSSinFreeStateWhenTTDisOccupiedandNoMAisIssued`). The other transitions are the purpose of `ComputeStatesOfVSSinFreeStateWhenTTDisOccupiedandTrainisLocatedandMAisIssued`.

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. It should be noted that the formal specification has been verified and validated using *Rodin* [3], an industrial-strength tool supporting the *Event-B* method [1]. The full specification can be found in [19]. 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 goal diagram.

3.1 The Root Level

Figure 6 represents the domain model associated with the root level of the SysML/KAOS goal diagram of Figure 4. 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[4]. Thus, $dom(front) \setminus dom(rear)$ represents the set of trains equipped with a ERTMS and not equipped with a TIMS.

Figure 7 represents the B System model raised from the translation of the root level of the goal diagram of Figure 4 and of the associated domain model of Figure 6. The domain model gives rise to sets, constants,

[4] 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

```
domain model ertms_etcs_case_study {
	concepts:
		concept TRAIN {
			is variable: false
		}

	deduced concepts:
		attribute domain dom(connectedTrain) attribute connectedTrain
		attribute domain dom(rear) attribute rear

	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:
		axm2: a<b
		axm3: TRACK=a..b
		inv4: !tr. (tr : dom(rear) => rear(tr) < front(tr))

}
```

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

SYSTEM ertms_etcs_case_study
SETS TRAIN
CONSTANTS a b TRACK
PROPERTIES
axm1: $\{a, b\} \subseteq \mathbb{N}$
axm2: $a < b$
axm3: $TRACK = a \,..\, b$
VARIABLES connectedTrain front rear
INVARIANT
inv1: $connectedTrain \in TRAIN \nrightarrow BOOL$
inv2: $front \in dom(connectedTrain) \rightarrow TRACK$
inv3: $rear \in dom(connectedTrain) \nrightarrow TRACK$
inv4: $\forall tr \cdot (tr \in dom(rear) \Rightarrow rear(tr) < front(tr))$

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: $rear := (\{TRUE \mapsto rear \lhd\!\!- \{tr \mapsto rear(tr) + len\}, FALSE \mapsto rear\})(bool(tr \in dom(rear)))$
END
END

Fig. 7. B System specification of the root level of the goal diagram of Figure 4

properties, variables and invariants of the formal specification. Predicates involving variables give rise to invariants and the others to properties. The *isFunctional* and *isTotal* indicators, appearing in attributes, are

used to guess if an attribute gives rise to 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`).

3.2 The First Refinement Level

```
domain model ertms_etcs_case_study_ref_1 parent domain model ertms_etcs_case_study {
        deduced concepts:
                attribute domain dom(MA) attribute MA
                data set power POW(TRACK) data set TRACK

        attributes:
                attribute MA domain: dom(connectedTrain) range: POW(TRACK) {
                        is variable: true  is functional: true   is total: false
                }

        predicates:
                inv2: !tr. (tr : dom(MA) => #p,q.(p..q<:TRACK & p<=q & MA(tr)=p..q)))
                inv3: !tr. (tr : dom(MA) => (front(tr) : MA(tr)))
                inv4: !tr. (tr : dom(rear) & tr : dom(MA) => rear(tr) : MA(tr))
                inv5: !tr1,tr2. ((tr1 : dom(MA) & tr2 : dom(MA) & tr1 /= tr2)=>MA(tr1) /\ MA(tr2)={})

}
```

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

Figure 8 represents the domain model associated with the first refinement level of the SysML/KAOS goal diagram of Figure 4. 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 (`inv2`), containing the train (`inv3` and `inv4`). Finally, the predicate `inv5` asserts that the MA assigned to two different trains must be disjoint.

Figure 9 represents the B System model raised from the translation of the first refinement level of the goal diagram of Figure 4 and of the associated domain model of Figure 8. 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`). This new 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 `sysml_kaos_po1`, `sysml_kaos_po2`, `sysml_kaos_po3` and `sysml_kaos_po4` represent the proof obligations related to the usage of the *MILESTONE* operator between the root and the first refinement levels: for an event `G`, `G_Guard` represents the guards of G and `G_Post` represents the post condition of its actions. Since each proof obligation has been modeled as an Event-B theorem, it has been proved based on system properties and invariants.

The above theorems are sufficient to prove the *milestone* refinement, but they are too restrictive. It may be impossible to discharge them while the refinement is correct [12]. The necessary and sufficient conditions to prove the milestone refinement are expressed by using operators of the temporal logic in the following way, for an abstract goal G milestone-refined into subgoals *G1* and *G2*:

(1) $G1\_Guard \Rightarrow G\_Guard$

(2) $\Box(G1\_Post \Rightarrow \Diamond G2\_Guard)$: each system state, corresponding to the post condition of *G1*, must be followed, at least once in the future, by a system state enabling *G2*.

(3) $G2\_Post \Rightarrow G\_Post$

REFINEMENT ertms_etcs_case_study_ref_1
REFINES ertms_etcs_case_study
VARIABLES connectedTrain front rear MA MAtemp
INVARIANT
inv1: $MA \in dom(connectedTrain) \pfun \mathbb{P}(TRACK)$
inv2: $\forall tr \cdot (tr \in dom(MA) \Rightarrow (\exists p, q \cdot (p \upto q \subseteq TRACK \land p \leq q \land MA(tr) = p \upto q)))$
inv3: $\forall tr \cdot (tr \in dom(MA) \Rightarrow front(tr) \in MA(tr))$
inv4: $\forall tr \cdot (tr \in dom(rear) \cap dom(MA) \Rightarrow rear(tr) \in MA(tr))$
inv5: $\forall tr1, tr2 \cdot ((\{tr1, tr2\} \subseteq dom(MA) \land tr1 \neq tr2) \Rightarrow MA(tr1) \cap MA(tr2) = \emptyset)$
inv6: $MAtemp \in dom(connectedTrain) \pfun \mathbb{P}(TRACK)$
inv7: $\forall tr \cdot (tr \in dom(MAtemp) \Rightarrow (\exists p, q \cdot (p \upto q \subseteq TRACK \land p \leq q \land MAtemp(tr) = p \upto q)))$
theorem sysml_kaos_po1: $ComputeTrainMA\_Guard \Rightarrow MoveTrainOnTrack\_Guard$
theorem sysml_kaos_po2: $ComputeTrainMA\_Post \Rightarrow AssignMAtoTrain\_Guard$
theorem sysml_kaos_po3: $AssignMAtoTrain\_Post \Rightarrow MoveTrainFollowingItsMA\_Guard$
theorem sysml_kaos_po4: $MoveTrainFollowingItsMA\_Post \Rightarrow MoveTrainOnTrack\_Post$

Event

ComputeTrainMA $\widehat{=}$
any tr p q len
where
grd1: $tr \in connectedTrain^{-1}[\{TRUE\}]$
grd2: $p \upto q \subseteq TRACK \land p \leq q$
grd3: $front(tr) \in p \upto q$
grd4: $tr \in dom(rear) \Rightarrow rear(tr) \in p \upto q$
grd5: $p \upto q \cap union(ran(\{tr\} \domsub MA)) = \emptyset$
grd6: $len \in \mathbb{N}_1$
grd7: $front(tr) + len \in TRACK$
then
act1: $MAtemp(tr) := p \upto q$
END

AssignMAtoTrain $\widehat{=}$
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 $\widehat{=}$
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: $front(tr) := front(tr) + len$
act1: $rear := (\{TRUE \mapsto rear \rhd\!\!\!\!- \{tr \mapsto rear(tr) + len\}, FALSE \mapsto rear\})(bool(tr \in dom(rear)))$
END
END

Fig. 9. B System specification of the first refinement level of the goal diagram of Figure 4

We have used the proof obligation $G1\_Post \Rightarrow G2\_Guard$ instead of (2), since $(G1\_Post \Rightarrow G2\_Guard) \Rightarrow (\Box(G1\_Post \Rightarrow \Diamond G2\_Guard))$, to deal with the fact that Event-B does not currently support the temporal logic.

3.3 The Second Refinement Level

Figure 10 represents the domain model associated with the second refinement level of the diagram of Figure 4. 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 concepts. The predicates `axm1..axm8` define each TTD as a contiguous part of the track and each VSS as a contiguous part of a TTD. The predicates `inv3` and `inv4` 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 $front(tr) \in ttd$ (`inv3`) or if tr is equipped with a TIMS ($tr \in dom(rear)$) and $(rear(tr)..front(tr)) \cap ttd \neq \emptyset$ (`inv4`). Finally, the predicates `inv5..inv7` 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 (`inv5`); 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` (`inv6`); if none of them is an INTEGER train, then they must be in two distincts TTD (`inv7`).

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 10, two new events (`ComputeTrainMAFollowingTTDStates`, `ComputeTrainMAFollowingVSSStates`), an extension of the event `MoveTrainFollowingItsMA` taking into

```
domain model ertms_etcs_case_study_ref_2 parent domain model ertms_etcs_case_study_ref_1 {
       concepts:
              concept TTD {                                          concept VSS {
                     is variable: false                                     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 type : STRING
                     data value FREE type : STRING
                     data value UNKNOWN type : STRING
                     data value AMBIGUOUS type : STRING
  }

  enumerated data set TTD_STATES {
              elements :
                     data value OCCUPIED type : STRING
                     data value FREE type : STRING
  }

       predicates:
              axm1: TTD <: POW1(TRACK)
              axm2: union(TTD) = TRACK
              axm3: inter(TTD) = {}
              axm4: !ttd. (ttd : TTD => #p,q.(p..q<:TRACK & p<q & ttd=p..q)))
              axm5: VSS <: POW1(TRACK)
              axm6: union(VSS) = TRACK
              axm7: inter(VSS) = {}
              axm8: !vss. (vss : VSS => #p,q,ttd.(ttd : TTD & p..q<:ttd & p<q & vss=p..q)))
              inv3: !ttd,tr. ( tr : dom(front) \ dom(rear) & ttd : TTD & front(tr) : ttd)
                      => (( ttd |-> OCCUPIED ) : stateTTD)
              inv4: !ttd,tr. (tr : dom(rear) & ttd : TTD & (rear(tr)..front(tr))/\ttd /= {})
                     => (( ttd |-> OCCUPIED ) : stateTTD)
              inv5: !tr1,tr2. (tr1 : dom(rear) & tr2 : dom(rear) & tr1 /= tr2)
                     => ( (rear(tr1)..front(tr1))/\(rear(tr2)..front(tr2))={})
              inv6: !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))
              inv7: !tr1,tr2,ttd. ( tr1 : dom(front)\dom(rear) & tr2 : dom(front)\dom(rear) &
                     tr1 /= tr2 & ttd : TTD & front(tr1) : ttd) => ( front(tr2) /: ttd)
}
```

Fig. 10. SysML/KAOS domain modeling of the goal diagram second refinement level

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. The specification below (Figure 11) represents the new definition of `MoveTrainFollowingItsMA` and the theorems related to the refinement operator. The

parameter `ttds` is introduced to capture the TTD requiring an update of their states because of the train movement (`grd4`, `grd5` and `act3`). Guards `grd6..grd9` ensure that the train movement will not lead to the violation of the safety invariants `inv5..inv7` : `grd6` stands for `inv5`; `grd7` and `grd8` stand for `inv6`; `grd9` stands for `inv7`.

REFINEMENT ertms_etcs_case_study_ref_2
REFINES ertms_etcs_case_study_ref_1
INVARIANT
theorem sysml_kaos_po1: $ComputeTrainMAFollowingTTDStates\_Guard \Rightarrow ComputeTrainMA\_Guard$
theorem sysml_kaos_po2: $ComputeTrainMAFollowingVSSStates\_Guard \Rightarrow ComputeTrainMA\_Guard$
theorem sysml_kaos_po3: $ComputeTrainMAFollowingTTDStates\_Post \Rightarrow ComputeTrainMA\_Post$
theorem sysml_kaos_po4: $ComputeTrainMAFollowingVSSStates\_Post \Rightarrow ComputeTrainMA\_Post$
theorem sysml_kaos_po5: $ComputeTrainMAFollowingTTDStates\_Post \Rightarrow not(ComputeTrainMAFollowingVSSStates\_Guard)$
theorem sysml_kaos_po6: $ComputeTrainMAFollowingVSSStates\_Post \Rightarrow not(ComputeTrainMAFollowingTTDStates\_Guard)$
Event MoveTrainFollowingItsMA $\widehat{=}$
any tr len ttds
where
grd1: $tr \in connectedTrain^{-1}[\{TRUE\}] \cap dom(MA)$
grd2: $len \in \mathbb{N}_1$
grd3: $front(tr) + len \in MA(tr)$
grd4: $ttds \subseteq stateTTD^{-1}[\{FREE\}]$
grd5: $\forall ttd \cdot (ttd \in stateTTD^{-1}[\{FREE\}] \wedge ((front(tr) + len \in ttd) \vee (tr \in dom(rear) \wedge ((rear(tr) + len \mathinner{..} front(tr) + len) \cap ttd \neq \emptyset))) \Rightarrow ttd \in ttds)$
grd6: $tr \in dom(rear) \Rightarrow (\forall tr1 \cdot ((tr1 \in dom(rear) \wedge tr1 \neq tr) \Rightarrow (rear(tr1) \mathinner{..} front(tr1)) \cap (rear(tr) + len \mathinner{..} front(tr) + len) = \emptyset))$
grd7: $tr \in dom(rear) \Rightarrow (\forall tr1, ttd \cdot ((tr1 \in dom(front) \setminus dom(rear) \wedge tr1 \neq tr \wedge ttd \in TTD \wedge front(tr1) \in ttd \wedge rear(tr) .. front(tr) \cap ttd \neq \emptyset) \Rightarrow front(tr1) < rear(tr) + len))$
grd8: $tr \in dom(front) \setminus dom(rear) \Rightarrow (\forall tr1, ttd \cdot ((tr1 \in dom(rear) \wedge tr1 \neq tr \wedge ttd \in TTD \wedge front(tr) + len \in ttd \wedge rear(tr1) .. front(tr1) \cap ttd \neq \emptyset) \Rightarrow front(tr) + len < rear(tr1)))$
grd9: $tr \in dom(front) \setminus dom(rear) \Rightarrow (\forall tr1, ttd \cdot ((tr1 \in dom(front) \setminus dom(rear) \wedge tr1 \neq tr \wedge ttd \in TTD \wedge front(tr1) \in ttd) \Rightarrow front(tr) + len \notin ttd))$
then
act1: $front(tr) := front(tr) + len$
act2: $rear := (\{TRUE \mapsto rear \leftarrow \{tr \mapsto rear(tr) + len\}, FALSE \mapsto rear\})(bool(tr \in dom(rear)))$
act3: $stateTTD := stateTTD \leftarrow (ttds \times \{OCCUPIED\})$
END

END

Fig. 11. B System specification of the second refinement level of the diagram of Figure 4

3.4 The Fifth Refinement Level

For the fifth refinement level, corresponding to the first refinement level of the goal diagram of Figure 5, the B System specification introduces four events raised from the translation of the goals and five theorems representing the proof obligations related to the usage of the *AND* operator between the fourth and the fifth refinement levels. These theorems are :

theorem sysml_kaos_po1: $ComputeStatesOfVSSinUnknownState\_Guard \Rightarrow ComputeVSSStatesFollowingTTDStates\_Guard$
theorem sysml_kaos_po2: $ComputeStatesOfVSSinOccupiedState\_Guard \Rightarrow ComputeVSSStatesFollowingTTDStates\_Guard$
theorem sysml_kaos_po3: $ComputeStatesOfVSSinAmbiguousState\_Guard \Rightarrow ComputeVSSStatesFollowingTTDStates\_Guard$
theorem sysml_kaos_po4: $ComputeStatesOfVSSinFreeState\_Guard \Rightarrow ComputeVSSStatesFollowingTTDStates\_Guard$
theorem sysml_kaos_po1: $ComputeStatesOfVSSinUnknownState\_Post \wedge ComputeStatesOfVSSinOccupiedState\_Post \wedge ComputeStatesOfVSSinAmbiguousState\_Post \wedge ComputeStatesOfVSSinFreeState\_Post \Rightarrow ComputeVSSStatesFollowingTTDStates\_Post$

4 Discussion

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 structure and the system requirements. 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 to stakeholders, even non-experts, and to better delineate the system boundaries. Using Rodin [3], we have formally verified and validated the safety invariants and the SysML/KAOS goal diagram refinement hierarchy.

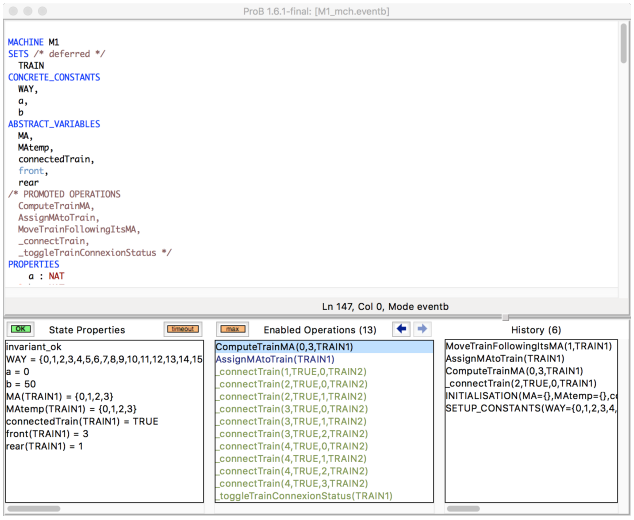

Fig. 12. Overview of the animation of the formal specification through proB

Through proB [10], we have animated the formal model (Figure 12). The full specification can be found in [19]. The work done on this case study allowed us to enrich the expressivity of the SysML/KAOS modeling languages. For example, we have introduced the property isTotal in the Attribute class of the domain modeling metamodel because an attribute may be partial and translated into a partial function. We have also introduced the *deduced concepts* in the domain modeling language and tool [20] that allow the representation and the manipulation of elements such as dom(AS) for the domain of the relation/attribute *AS*, ran(AS) for the range of *AS* or POW(DS) for the powerset of the dataset *DS*, that do not appear in formalisms as *OWL* [17] or *PLIB* [16].

Nevertheless, 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 proof obligations associated with the SysML/KAOS refinement operators used.

Table 2. Key Characteristics related to proof of SysML/KAOS refinements

Characteristics	**L0/L1**	**L1/L2**	**L2/L3**	**L3/L4**	**L4/L5**	**L5/L6**
Invariants	**4**	**6**	**3**	**6**	**5**	**5**
Proof Obligations (PO)	**8**	**12**	**6**	**12**	**5**	**5**
Automatically Discharged POs	**7**	**2**	**5**	**3**	**0**	**0**
Interactively Discharged POs	**1**	**10**	**1**	**9**	**5**	**5**

5 Conclusion and Future Work

This paper was focused on the usage 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 implementation [8, 5, 14]. Translation rules, supported by tools [13, 20], 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]. The work done also allowed us to enrich the expressiveness of the SysML/KAOS modeling languages.

Work in progress is aimed 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* [15] 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 [FP6 IST-511599 RODIN project], Lecture Notes in Computer Science, 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.: Une extension SysML pour l'ingénierie des exigences dirigée par les buts. In: 28e Congrès INFORSID, France. pp. 277–292 (2010)
7. Gnaho, C., Semmak, F., Laleau, R.: Modeling the impact of non-functional requirements on functional requirements. In: Parsons, J., Chiu, D.K.W. (eds.) Advances in Conceptual Modeling - ER 2013 Workshops, LSAWM, MoBiD, RIGiM, SeCoGIS, WISM, DaSeM, SCME, and PhD Symposium, Hong Kong, China, November 11-13, 2013, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8697, pp. 59–67. Springer (2013), https://doi.org/10.1007/978-3-319-14139-8_8
8. Hoang, T.S., Butler, M., Reichl, K.: The Hybrid ERTMS/ETCS Level 3 Case Study. ABZ pp. 1–3 (2018), http://users.ecs.soton.ac.uk/asf08r/ABZ2018/ERTMS_L3_Hybrid.pdf
9. van Lamsweerde, A.: Requirements Engineering - From System Goals to UML Models to Software Specifications. Wiley (2009)
10. Leuschel, M., Butler, M.J.: Prob: A model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003: Formal Methods, International Symposium of Formal Methods Europe, Pisa, Italy, September 8-14, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2805, pp. 855–874. Springer (2003)
11. 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 (2016)
12. 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. IEEE Computer Society (2011)
13. Matoussi, A., Laleau, R.: Un outil de construction de spécifications abstraites event-b dirigée par les buts. Ingénierie des Systèmes d'Information 16(5), 143–166 (2011), https://doi.org/10.3166/isi.16.5.143-166
14. Nicola, F., Henri, v.H., Laura, A., Maarten, B.: ERTMS Level 3: the Game-Changer. IRSE News View p. 232 (Apr 2017)
15. Openflexo: Openflexo project (2015), http://www.openflexo.org
16. 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)
17. Sengupta, K., Hitzler, P.: Web ontology language (OWL). In: Encyclopedia of Social Network Analysis and Mining, pp. 2374–2378 (2014)
18. Tueno, S.: Event-B Specification of Translation Rules (2017), https://github.com/stuenofotso/SysML_KAOS_Domain_Model_Parser/tree/master/SysMLKAOSDomainModelRules_ordered_attempt2
19. Tueno, S.: SysML/KAOS Domain Modeling Approach on ERTMS/ETCS Level 3 Hybrid (2017), https://github.com/stuenofotso/SysML_KAOS_Domain_Model_Parser/tree/master/ABZ18_ERTMS
20. Tueno, S.: SysML/KAOS Domain Modeling Tool (2017), https://github.com/stuenofotso/SysML_KAOS_Domain_Model_Parser
21. Tueno, S., Laleau, R., Mammar, A., Frappier, M.: Formal Representation of SysML/KAOS Domain Model (Complete Version). ArXiv e-prints, cs.SE, 1712.07406 (Dec 2017)
22. Tueno, S., Laleau, R., Mammar, A., Frappier, M.: The SysML/KAOS Domain Modeling Approach. ArXiv e-prints, cs.SE, 1710.00903 (Sep 2017)
23. 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 (2017)