Modeling the Hybrid ERTMS/ETCS Level 3 Implementation through Goal Diagrams and Ontologies Using the FORMOSE Approach

Steve Jeffrey Tueno Fotso¹,², Marc Frappier¹, Régine Laleau², and Amel Mammar³

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].

![Excerpt from a metamodel of the B System specification language](image)

**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]: \( \text{if CurrentCondition then sooner-or-later TargetCondition} \). 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
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

<table>
<thead>
<tr>
<th>Translation Of</th>
<th>Domain Model</th>
<th>B System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract domain model</td>
<td>DM ∈ DomainModel</td>
<td><em>o-DM</em> ∈ System</td>
</tr>
<tr>
<td>Domain model with parent</td>
<td>DM → PDM</td>
<td><em>o-DM</em> ∈ Refined <em>o-DM</em> refines <em>o-PDM</em></td>
</tr>
<tr>
<td>Abstract concept</td>
<td>DM is associated to PDM through the parent association PDM has already been translated</td>
<td><em>o-CO</em> ∈ AbstractSet</td>
</tr>
<tr>
<td>Concept with parent</td>
<td>CO ∈ Concept</td>
<td><em>o-CO</em> ∈ Concept</td>
</tr>
<tr>
<td></td>
<td>CO is not associated to a parent concept</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CO is associated to PCO through the parent concept association PCO has already been translated</td>
<td><em>o-CO</em> ∈ Constant _o-CO ⊊ <em>o-PCO</em></td>
</tr>
</tbody>
</table>
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.

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.

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
the current VSS states can be split 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, \(\text{dom(front)} \setminus \text{dom(rear)}\) represents the set of trains equipped with an 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, constants, operations, and variables used in the B System model. The domain model is used to define the initial state of the system and the set of valid transitions. The B System model is then used to specify the behavior of the system and to verify its correctness.

\(^4\) 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 \Rightarrow BOOL
  inv2: front \in dom(connectedTrain) \Rightarrow TRACK
  inv3: rear \in dom(connectedTrain) \Rightarrow TRACK
  inv4: \forall tr.(tr \in dom(rear) \Rightarrow rear(tr) < front(tr))

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 (\(g1d\)) 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 (\(g1d3\)).

### 3.2 The First Refinement Level

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

  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 \(MA\) temp (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
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 TMS ($tr \in dom(rear)$ and $(rear(tr)..front(tr)) \cap ttd = \emptyset$ (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 TMS 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

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

\textbf{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:

\begin{align*}
\text{theorem sysml\_kaos\_p01:} & \text{ ComputeTrainMAFollowingTTDStates\_Guard } \Rightarrow \text{ ComputeTrainMA\_Guard} \\
\text{theorem sysml\_kaos\_p02:} & \text{ ComputeTrainMAFollowingVSSStates\_Guard } \Rightarrow \text{ ComputeTrainMA\_Guard} \\
\text{theorem sysml\_kaos\_p03:} & \text{ ComputeTrainMAFollowingTTDStates\_Post } \Rightarrow \text{ ComputeTrainMA\_Post} \\
\text{theorem sysml\_kaos\_p04:} & \text{ ComputeTrainMAFollowingVSSStates\_Post } \Rightarrow \text{ ComputeTrainMA\_Post} \\
\text{theorem sysml\_kaos\_p05:} & \text{ ComputeTrainMAFollowingTTDStates\_Post } \Rightarrow \text{ not(ComputeTrainMAFollowingTTDStates\_Guard)} \\
\text{ theorem sysml\_kaos\_p06:} & \text{ ComputeTrainMAFollowingVSSStates\_Post } \Rightarrow \text{ not(ComputeTrainMAFollowingTTDStates\_Guard)}
\end{align*}

\textbf{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 bridged 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.
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>
<thead>
<tr>
<th>Characteristics</th>
<th>L0/L1</th>
<th>L1/L2</th>
<th>L2/L3</th>
<th>L3/L4</th>
<th>L4/L5</th>
<th>L5/L6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invariants</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Proof Obligations (PO)</td>
<td>8</td>
<td>12</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Automatically Discharged POs</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>12</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Interactively Discharged POs</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>9</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

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

2. ANR-14-CE28-0009: Formose ANR project (2017)